新版 図解

Z-80の使い方

横田 英一 著

オーム社

新版 図解

Z-80の使い方

横田 英一 著

オーム社

は し が き

「マイクロコンピュータ Z-80 の使い方」は，ちょうど世の中にマイクロコンピュータが普及しはじめ，エレクトロニクスエンジニアの新しいツールとして脚光を浴びはじめた時期に刊行し，なかでも本流となった Z-80 の解りやすい入門書として，10 万を越す読者を得，この本で育った先輩から，後輩へ受け継がれております．このような当初予想もしなかった反響に驚くとともに，強い責任をも感じている次第です．

エレクトロニクスの進化の速度は級数的に増し，マイクロコンピュータの世界も，16 ビットから 32 ビットへと広がり，要求と可能性はエスカレートし，RISC，並列処理などの新しいキーワードによる夢は限りなく拡大しております．しかし，いかに高性能なマイクロコンピュータといえども，ノイマン型であるかぎり基本は同じで，スキルの習得には一定のアプローチルールがあるように思えます．

8 ビットマイコンは汎用性にすぐれており，機能的によく整理され，性能において標準的な位置にあります．教材の入手も容易でマイクロコンピュータを習得するうえでのステップとしては，手頃なものです．

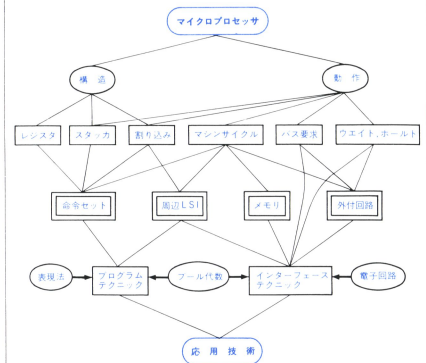
その 8 ビットマイコンの中でも Z-80 は，世界標準と呼ぶに相応しく，オリジナルソースであるザイログ社にとどまらず，国内主要メーカ各社からも Z-80 をベースにした発展型が発表され，好評を博しております．Z-80 の基本設計のよさが認められた結果，多くのエレクトロニクスエンジニアが Z-80 を必要条件と考え習得するに至ったため，以降の 8 ビットマイコンはこれに準拠することが市場に受け入れられる最短距離と，だれしもが認めるところとなったのです．

は　し　が　き

新たな展開の時期に至り，Z-80を踏み台にして飛び立って行ったマイクロコンピュータ自身の軌跡を，これからマイクロコンピュータを学ぼうと志す若いエンジニアの指針とするべく，本書の内容も，より適切な形に改めることにいたしました．次の世代のエレクトロニクスエンジニアの必携ツールである，マイクロコンピュータ応用技術の第一歩を本書から読み取っていただければ望外の幸いです．

1993年7月

著者しるす

マイクロプロセッサの応用技術

基礎知識

- 表現法
  （用語）
  （16進表現）
  （アセンブリ言語）
- ブール代数
  （真理値表）
  （論理回路）
- 電子回路
  （オームの法則）
  （電子部品）
  （半導体）

付加事項

- 経験と知識に基づく　慣れ≒ヤマカン　を養うこと．
  自分でやってみることが重要
- 中身がどうなっているかは問題ではない．
  どうすればどうなるかを知って使いこなすことが重要．

学習のポイント

マイクロコンピュータは，ハード面，ソフト面が有機的に結合しあいシステムを形成しています．ページ展開に従った一次元的な説明では到底表現しきれないものです．平面的，二次元的つながりを重要視する意味で項目間の参照は不可欠です．また，一読して理解できないことを嘆くなら再読を勧めます．一回目ではぼんやりと全体像をとらえ，2回目では後ろに書いてあることを思い起こしながら前の説明を見て行きます．次に自分でプログラムを考えてみて下さい．例題のプログラムを自分なりに手直しするだけでも良いのです．自分自身で試行することにより不明点をとらえ，解説を読み直せば確実に身につきます．

（1） マイクロコンピュータは物性や自然科学とは異なり，人間が考え出して作ったものです．使い易く，覚え易くするためにさまざまな知恵を絞ってあります．マイクロコンピュータを理解するのは，作られたルールを習得することととらえ，自然科学の探求とは異なった考え方で対処すればいたって気軽に取り組めると思います．

（2） また，マニュアルのような正確緻密な表現よりも，平易な解説を心がけました．コンピュータを理解できないという人のほとんどが，聞き慣れない用語とアルファベットの羅列に面食らっているようで，この点も読み進むうちにだんだんとなじめるように配慮したつもりです．

（3） ただ，マイクロコンピュータを学習するうえで，オームの法則程度の電子回路，ブール代数，論理回路（ロジックIC）等についての最低限の知識は必要です．この点は他に良書が多数あることを理由に，触れる程度に止めました．

（4） マイクロコンピュータを学習する上で，まず『何をしたいのか』といった具体的テーマをもち，『何ができるのか』からのフィードバックにより回答を模索して行くことが，一番の近道であると思います．この作業は物作りすべてにあてはまる事であり，エンジニアリングの本質でもあると思います．

目　　次

Z-80学習用に最適なポケットコンピュータ　PC-E200(シャープ)

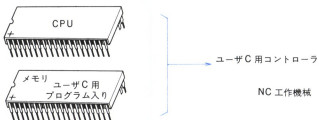
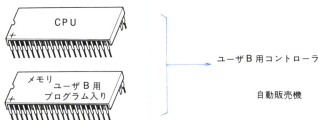
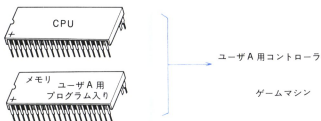
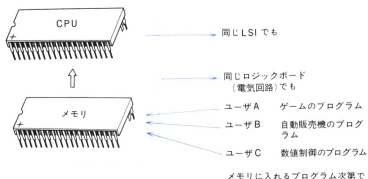
1

マイクロプロセッサの特質

コンピュータという機械が発明された目的は，人間には不可能な複雑な計算を短時間に行うことでした．たとえば円周率の計算を一生の大半を費やして行っても数百桁だったのに，コンピュータなら瞬時にもっと多くの桁を間違いなくやってのけることができます．人間を月へ送り込むのも，正確な軌道計算が刻々と得られるためにできたことで，コンピュータなしでは考えられないのです．ある手順の決まった仕事を，高速かつ正確に処理することのできる機械，これがコンピュータの目的であり，その目的は十分に達せられたといえるでしょう．

ところが，ストアドプログラム方式すなわち，処理手順を示すプログラムを，コンピュータに入力すべき情報の一つとして扱うことのできる今日のコンピュータ方式では，当初の目的以外に，大きな特徴が認識されました．全く同一の機械（ハードウェア）に対し，使用者が作成するプログラム（ソフトウェア；これは使用者によってそれぞれに異なる）をメモリに入れることにより，使用者のそれぞれの目的に合った仕事をする，という特徴です．コンピュータメーカは同じ機械を大量に生産すれば，科学計算であろうと，在庫管理であろうと，人事管理，生産ラインコントロール，座席予約，相性判断まで，プログラム次第で適用されてしまうのです．

半導体の技術が進歩して数千，数万の部品を数ミリ四方の中へ作り込めるようになったとき，専用の機能を持ったLSIが，さまざまな目的に合わせて作られました．ところが大量生産にしか向かないLSIの弱点を補うべく，半導体メーカは汎用LSIの思想をコンピュータに見出したのです．コンピュータをLSI化すれば，多く用途のある，つまり大量に売れるLSIが作れると考えたのです．マイクロプロセッサはこのような中から誕生し，プログラムによって機能が決定される汎用の論理素子として，すばらしい発展をとげ，現在も発展しつつある，“電子部品”なのです．マイクロプロセッサを中心としたマイコンシステムを利用しようとする場合，この辺を的確に把握してかかることが重要です．

CPU
同じLSIでも
同じロジックボード
（電気回路）でも
メモリ
ユーザA　ゲームのプログラム
ユーザB　自動販売機のプログラム
ユーザC　数値制御のプログラム
メモリに入れるプログラム次第で
CPU
メモリ　ユーザA用
プログラム入り
ユーザA用コントローラ
ゲームマシン
CPU
メモリ　ユーザB用
プログラム入り
ユーザB用コントローラ
自動販売機
CPU
メモリ　ユーザC用
プログラム入り
ユーザC用コントローラ
NC工作機械

2 マイコンシステムの種類

マイコンと呼ばれるコンピュータシステムには，いくつか考えられます．一つに，パーソナルコンピュータやオフィスコンピュータに近い性能を持つものを**“マイコン”**と呼びます．これらは，コンピュータを小さく，安価にまとめ，事務所や大学の研究室や個人の知的玩具としてまで普及した個人用コンピュータシステムです．電気の知識がなくても，簡単な言葉を覚えれば誰でも使えるテレビ，オーディオなどと肩を並べる全く新しいマニアライクな道具といえます．この場合のマイコンは，**“My Computer”**と解釈すべきでしょう．

もう一つの“マイコン”は，パソコンほどのはなばなしさはありませんが，工作機械や自動販売機や家電製品に組み込まれている，機器制御用の**ワンボードマイコン**です．基板上に作られたマイクロコンピュータシステムは，産業用としての中心をなすものです．各種の用途でそれぞれの目的に合った設計がなされ，特徴を出しています．組み込まれた時点では，プログラムは固定化され，機器のユーザには，コンピュータとしての使用はできないのが普通です．

1個のLSI上にメモリや入出力ポートを作り込んだ**ワンチップマイコン**は，時計や電卓に使われています．中身を分析すれば，マイクロコンピュータに違いありませんが，LSIを外面から見た場合，一つの機能を持った専用LSIになってしまいます．開発過程では，コンピュータの特徴が活きて，短時間に安価な開発経費で専用LSIが作れるのです．機器制御用マイコンシステムをワンチップ化したと考えればよく，生産数量がきわめて大きい場合，コストやサイズ，消費電力などの点で，効果をあげることができます．

また，マイコンシステムを構成する最重要部品であるマイクロプロセッサ，つまりCPUのLSIを“マイコン”と呼びます．しかし，これだけではコンピュータとはいえず，周辺やプログラムを含めてコンピュータシステムが完成するのですから，あくまでCPUと呼ぶべきです．

Z 80 CPU
マイクロプロセッサ (CPU)
Z 80 PIO
入出力LSI
Z 80 CTC
カウンタタイマなど
周辺LSI
メモリ
メモリ
TTL
ロジックIC
抵抗
コンデンサなど
基板
コネクタ
など
電源
CRT
ブラウン管
プリンタ
など
キーボード
フロッピ装置
など
ワンボードマイコン(制御用)
ソフトウェア
ワンボードマイコンやパーソナルコンピュータはマイコンシステムの応用製品
パーソナルコンピュータ
一方，ワンチップマイコンは開発過程がコンピュータである，専用LSIである
プログラム
マイコン工場
電卓用LSI
電卓工場
液晶表示板
1234
電卓用LSI
電池
キーボード
1 2 3 C
4 5 6 √
7 8 9 %
+ − × ÷

3

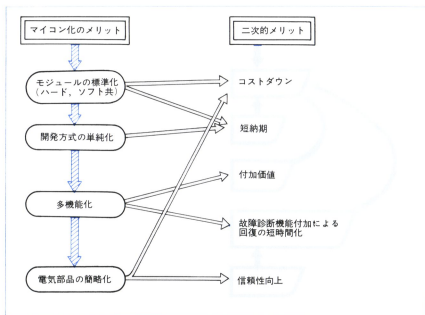
特質を活かすシステム設計

コンピュータは，仕事を高速に処理することと，同じ機械(ハードウェア）でもプログラム（ソフトウェア）を変えるだけで異なった働きをさせられること，の2点が最大の特徴です．制御用マイコンは，メカニズムやリレーや論理回路によるワイアードロジックからの延長線上にあり，高速性では，ワイアードロジックには一歩ゆずります．むしろ，プログラムにかかる機能面の柔軟性が大きなメリットです．また，多機能化することが比較的容易であり，故障時に自己診断をさせることすら可能です．マイコンボードとして決定された能力の限界以内であれば，同一の規格化された基板を何種類もの製品に応用することができ，開発コストの引き下げに効果が期待できるようになります．プログラムもモジュール化し保管をすることにより，経験の蓄積ができ開発期間の短縮が図れます．

マイコンシステムの設計にあたっては，ハードウェア（以下，ハードといいます）とソフトウェア（以下，ソフトといいます）の分担を決定することがポイントになります．回路設計とプログラム設計の前段階としてのシステム設計といわれる作業です．往々にして，電子技術者が設計したシステムはハードにウエイトがおかれてしまう傾向があります．ハード，ソフト共に精通した人が，目的の機能と，開発コスト，ランニングコスト，メンテナンスコストなどすべてにわたって検討して決定すべきです．特に部品やボード，プログラムの標準化という意味から，将来にわたって要求される機能の変動を見通すことが重要になります．変動要素はソフトの分担とし，メモリの交換だけで対応できなければなりません．

概してコンピュータの特質を考えるならば，タイミング的に不可欠な部分をハード化し，他はソフトで実現する方法に利があるといえます．ハードを最小限に止めることは，材料費，経年劣化，故障率を低減するうえで有利であることは明白です．

マイコン適用分野

- 多機能であって，ICレベルの論理素子で構成すると，大規模になりすぎるもの ——→ パーソナルコンピュータ
- 大同小異の機種が多く，いちいち論理素子で構成したのでは設計に手間がかかりすぎるもの ——→ 端末装置，NC 機器
- 納入先ごとの変更があるもの ——→ ビル防災システム，ホテル管理システム
- あとで変更のあり得るもの ——→ 自動販売機，料金計算機
- 開発時間の限られる場合 ——→ ハードウェアは標準品を使い，プログラムだけを入れ換える

必要条件

4 Z-80ファミリの特徴

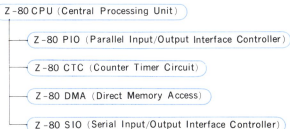
Z-80はインテル社が開発した8ビットマイコン8008からの進化の流れをくみ，8ビットプロセッサとして一応の完成を見た製品です．デビューのタイミングがマイコンの普及期と一致したこともあり，多くの技術者に受け入れられた，標準的な体系を備えた汎用プロセッサです．

5ボルト単一電源，単相クロック，16ビットの減算などの機能を持ち，発表当時としては極めて先進的なマイコンでした．さらに機能を充実した8ビットマイコンはZ-80以降も開発されています．しかし基本的な考え方は，Z-80を踏襲したものがほとんどです．すでに普及した基本体系を変えないことが，開発装置などの環境や技術者のノウハウをはじめとする蓄積資産の有効活用を可能にし，容易に市場展開ができるのです．

初期のZ-80ファミリは，N-MOSプロセスでしたが，その後より低消費電力を求めてC-MOSプロセスへ移行し，処理速度，動作温度範囲，動作電圧範囲，パッケージなどユーザの選択範囲を広げるべく，いろいろなバリエーションがそろっています．

機器の小型化の動きは複数のLSIを一つのLSIに集積することさえ要求し，コア方式といった単純化，画一化された開発手順で，用途別に特化されたIC「ASIC」を次々に生み出すことが可能になりました．Z-80 CPUを核とし周辺機能を取り入れたワンチップマイコンも各社から発表され，プログラム領域を外部におくことのできるタイプのものは，量産数の少ない用途でも高集積LSIのメリットを取り入れることができ，喜ばれています．組み込む周辺機能はいろいろな組み合わせがあり得，メーカでは各種の製品をそろえています．最近では，Z-80の弱点とされていた，乗除算命令などを追加したCPUコアもあり，またZ-80ファミリ以外のA/Dコンバータやオペアンプ，コンパレータなどもそろえて，ますます選択の幅が広がり，使いやすくなっています．

Z-80 ファミリ

- Z-80 CPU (Central Processing Unit)
  - Z-80 PIO (Parallel Input/Output Interface Controller)
  - Z-80 CTC (Counter Timer Circuit)
  - Z-80 DMA (Direct Memory Access)
  - Z-80 SIO (Serial Input/Output Interface Controller)

特　徴

- 8ビット標準アーキテクチャ
- 割り込み機能 ——→ コントローラ不要，プログラムが簡単
- レジスタ群 ——→ プログラム容量の圧縮・スピードアップ
- 命令セット ——→ プログラム容量の圧縮・スピードアップ
- リフレッシュ機能 →ダイナミックメモリ使用可能 → コストダウン

命令セットの特徴

- 8ビットマイコン標準的命令体系
- レジスタ，メモリのビットのセット，リセットテスト
- メモリブロックの転送，サーチ，入出力
- 2の補数をとる命令
- 4ビット単位のローテーション
- 整理されたニモニック（暗記用命令コード）

5 CPU コア ASIC

マイコンは大量生産の得意な半導体メーカと，少量生産でも大規模集積回路の恩恵を蒙りたい電子機器メーカのギャップを埋める，虹のかけ橋として登場したわけです．Z-80 ファミリでは個別の機能ユニットをそれぞれの LSI としてまとめ，ユーザは必要とするシステムを，LSI の組合わせで構築するという方法で，これを実現してきたのです．ところが，小ささへの価値観が高まるにつれて，複数 LSI の組合わせを 1 個の LSI に集積したいという要求が強くなり，比較的量産数の少ない分野でもワンチップマイコン導入の動きが起き，しかも応用分野ごとの異なった要求仕様を満足する必要が出てきました．半導体メーカは，画一的な製品構成から抜け出て，応用範囲を拡大すべく，それぞれのユーザの要求に合う製品をラインナップしたいところですが，これまでの大規模集積回路の開発手法では自ずと限界があり，対応を模索してきました．

ASIC（Application Specific IC）［エイシック］というのは，特定用途向け IC と訳され，適応範囲を広げることを考慮せず，文字どおり特定用途だけを対象に考えて，LSI を開発する手法あるいはその製品のことで，新たな要望仕様に対するカスタム LSI を比較的容易に実現できるものです．個別の LSI に相当する機能セルを，それぞれの応用に最適な組合わせで選択し，一つの LSI 上に作り込む方法で，システムオンチップとも呼ばれます．

CPU コア ASIC は CPU を核として必要な機能を組込んだ LSI で，あたかもプリント基板上に CPU と周辺 LSI を並べてできるワンボードコンピュータのように，シリコンチップ上に機能セルを並べてできるワンチップマイコンなのです．このCPU コアとして，最も普及している Z-80 CPU を使用できるメリットははかり知れず，すでに数社の半導体メーカから発表されています．

しかし，いかに ASIC といえどもある程度の数量規模がないと生産ラインに乗せることはできませんし，数量規模が見込めれば価格的に有利になります．半導体メーカでは数量規模がないユーザのために，いろいろな組合わせを持ったコアマイコンをあらかじめ用意し，対処しています．

システム構成に必要な LSI の組合せ

CTC

CPU

RAM

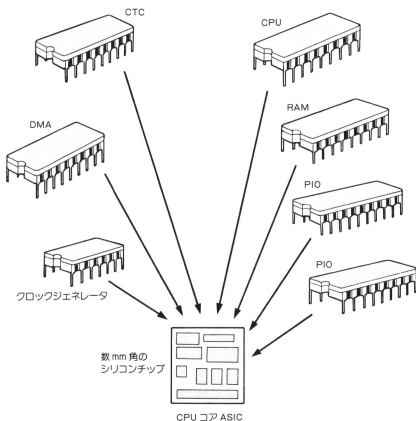
DMA

PIO

PIO

クロックジェネレータ

数 mm 角の
シリコンチップ

CPU コア ASIC

必要な機能を一つのシリコンチップ上に集積した IC

6

Z-80 CPU

CPU (Central Processing Unit) はZ-80ファミリの中核をなすものであり，ファミリの性格を決定する中心要素です．なお，ファミリでCPU以外のLSIを，周辺LSIとかペリフェラルと呼ぶことがあります．

Z-80 CPUには8ビットの汎用レジスタが12個あります．2個をつなげて16ビットレジスタとして使用することもできます．インデックスレジスタは2個あり，他にスタックポインタ，プログラムカウンタ，割り込みベクトルレジスタ，メモリリフレッシュレジスタ，フラグレジスタ，アキュムレータがあります．

これらのレジスタや命令を解析し実行する制御部や演算部の働きは，以下の項で解説します．

外部端子には，アドレスバス16本，データバス8本，システム制御線6本，CPU制御線5本，バス制御線2本，それにクロックと電源が出ています．図で，トライステートと記入されている端子は，“H”と“L”の二値状態のほかに，端子と内部が電気的に切り離された“ハイインピーダンス”状態を持つものです．動作と関係のないときはこの状態になっており，外部からの負荷にならないようになっています．

Z-80ファミリは，使用可能なクロック周波数により，いくつかのバージョンに分けられています．また，停止時に消費電流を減らす機能が付いたものや，特に高信頼度の要求される用途向けなど，さまざまなバリエーションのデバイスが作られています．

命令解析部
演算部
レジスタ群
制御部
データバス制御部
アドレスバス制御部

ϕクロック
V_{CC}（+5V）
GND

M1
MREQ Ⓣ
IORQ Ⓣ
RD Ⓣ
WR Ⓣ
RFSH Ⓣ
システム制御線

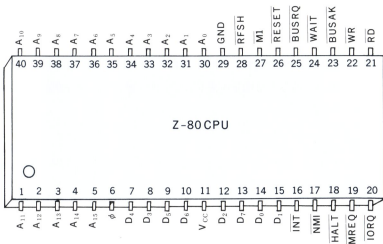
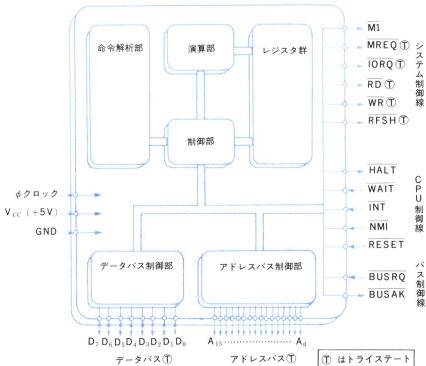
HALT
WAIT
INT
NMI
RESET
CPU制御線

BUSRQ
BUSAK
バス制御線

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
データバスⓉ

A_{15} …………………… A_0
アドレスバスⓉ

Ⓣ はトライステート

7

Z-80 PIO

CPU の次によく使われるのが，この **PIO**(Parallel Input/Output Interface Controller）です．CPU からの信号を受けて，外部の装置に出力したり，外部装置からの信号を受けて，CPU へ伝えたりするデバイスです．

CPU の信号は，データバスを入出力以外の用途にも時分割で共用しているため，大変複雑な信号になっています．PIO は，この中からある特定の時点のデータバス上の信号をとらえて外部との受け渡しをするためのもので，一般にラッチと呼ばれるロジックに，使いやすい機能を付け加えた LSI といえます．信号は 8 ビットを並列に扱います．動作は次の四つのモードがあります．

モード 0	出力モード	ハンドシェークにより 8 ビットデータを入・出力する．
モード 1	入力モード	
モード 2	入出力モード	
モード 3	ビットモード	（ビット単位に入力，出力を選択できる）

モードやその他の動作条件は，CPU からの信号により内部の制御レジスタに制御ワードを書くことによって設定されます．割り込みは CPU と PIO(または他のペリフェラル）のみで制御され，優先順位の決定や，どのペリフェラルからの割り込みかの解析や，割り込み処理プログラムルーチンからのリターンは，自動的に行われます．

PIO の内部にはほぼ同一のポートが二つあり，それぞれ別の動作モードで使うこともでき，割り込みも 2 系統発生させることができます．優先順位は，A ポート → B ポートの順です．

データバス
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7

データバス制御部

制御部
A ポートベクトル
B ポートベクトル
制御レジスタ
//

M1
RD
IORQ
CE
B/A
C/D

割込み
制御部

INT
IEI
IEO

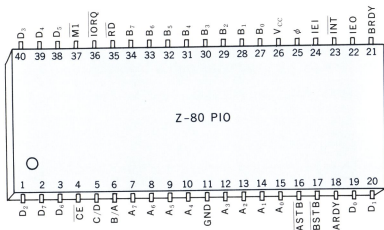
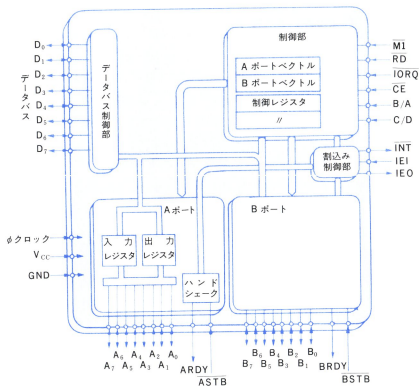
A ポート
入　力
レジスタ
出　力
レジスタ
ハンド
シェーク

B ポート

φクロック
V_{CC}
GND

A_6 A_4 A_2 A_0
A_7 A_5 A_3 A_1
ARDY
ASTB

B_6 B_4 B_2 B_0
B_7 B_5 B_3 B_1
BRDY
BSTB

8

Z-80 CTC

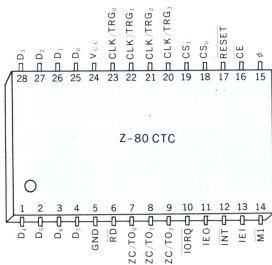
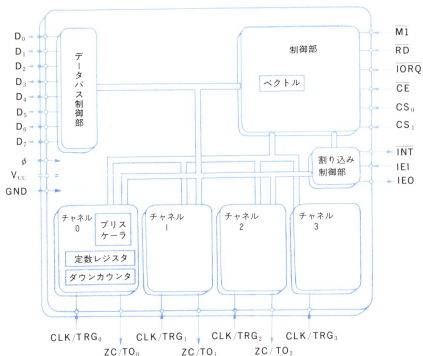
CTC（Counter Timer Circuit）は，パルスのカウントダウンをするLSIです．不特定周期の外部パルスをカウントし，設定数になると割り込みを入れたり，ゼロカウント出力を出したりすることができます．不特定なパルスでなくクロックのような決まった周期のパルスをカウントすることにより，タイマとしても使えるわけです．

何個目のパルスで割り込みまたはゼロカウントするかの設定や，カウントパルスのエッジ（立ち上りか立ち下りか）の選択などはCPUからの書き込みによってプログラムされます．またカウントの途中での残り数は，CPUがカウンタの内容を読み出すことによって知ることができます．

設定できる数は1〜256までですが，くり返して何回目かの割り込みでカウント終了することにより範囲は広がります．また，ゼロカウント出力を次チャネルのカウンタへ入れてやれば，256^2までが扱えます．したがって，n個つなげば256^nとなります．

1個のCTCには四つのチャネルが内蔵されていて，別々の目的に使用することができます．ただしピン数の関係で4個目のチャネルは，ゼロカウント/タイムアウトの出力が出ていませんので，割り込みによりカウント終了となる使い方しかできません．

システムクロックのカウントダウンによるタイマとしての使用時は，プリスケーラによってクロックを16または256分周したパルスをカウントします．タイマの起動は自動的に行うことも，また外部トリガによって行うこともできます．

D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
ϕ
V_{CC}
GND
データバス制御部
制御部
ベクトル
$\overline{M1}$
$\overline{RD}$
$\overline{IORQ}$
$\overline{CE}$
CS_0
CS_1
$\overline{INT}$
IEI
IEO
割り込み制御部
チャネル 0
プリスケーラ
定数レジスタ
ダウンカウンタ
チャネル 1
チャネル 2
チャネル 3
CLK/TRG_0 CLK/TRG_1 CLK/TRG_2 CLK/TRG_3
ZC/TO_0 ZC/TO_1 ZC/TO_2

9 Z-80 DMA

メモリ内，あるいはメモリと入出力（IO）ポートとのデータの転送は，通常はCPUが一度レジスタへ読み込み，次に書き出すという方式で行います．Z-80には，単命令で一連の複数のデータを転送するブロック転送命令があります．プログラムステップ数は少なくなり，便利ではありますが，転送に要する時間は，1バイトずつくり返しループを実行して，転送するのと大差ありません．この転送の作業だけを高速に行う周辺LSIが**DMA**（Direct Memory Access）です．

CPUから，あらかじめ転送されるデータの入った元のアドレス（ソース）と転送先のアドレス（ディスティネーション）と，転送するバイト数をDMA内のレジスタへ書き込めば，自動的に指定の転送を行います．ソースまたはディスティネーションは，自動的にカウントアップされますが，特定の入出力ポートへ次々に出力または入力するときはカウントアップを止めておくこともできます．

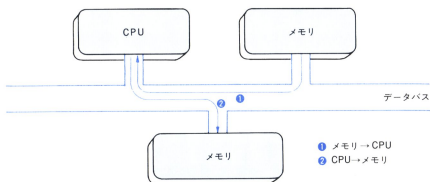
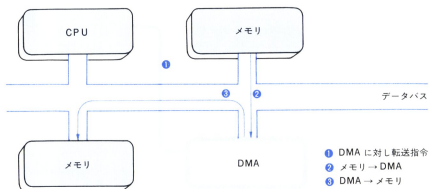
転送だけでなく，転送しながら，あるいは転送をせずに指定されたビットパターンとデータのビットパターンの一致をチェックすることもできます．これを**サーチ**と呼んでいます．サーチの終了（一致）は，転送終了とは区別できるよう割り込みがかかります．

DMAが動作中は，アドレスバスやデータバスは占有されますのでCPUは待ち状態になりますが，指定によりCPU優先として1バイト単位に転送することや，外部ロジックからの切り換えによりCPUへバスをあけ渡すこともできます．

DMAやSIOを使うシステムは，かなり大がかりなものになります．
本書では，初歩的な内容をわかりやすく解説することが主目的ですから，これらを対象からはずし，次のステップでの修得を期待します．

通常の転送（メモリ→メモリの場合）

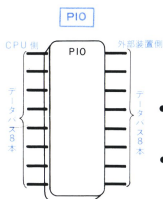
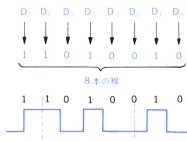
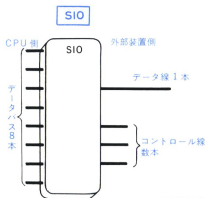
DMA 転送（メモリ→メモリの場合）

10

Z-80 SIO

PIOは，データを並列に入出力するためのポートコントローラですが，このSIO（Serial Input/Output Controller）は，データを直列に，すなわち時間経過に従って入出力するポートコントローラです．電話回線を使って長距離の通信を行なう場合はもとより，同一機器の匡体内でもケーブルが長くなる所は，データ通信線の数が少なくてすむシリアル伝送を利用することがあります．もちろん他の条件が同一なら伝送速度はパラレルの1/8になることはやむを得ません．

シリアル伝送の方式には，同期・非同期，バイト指向・ビット指向などがあり，また通信速度や電圧，電流など，何種かの規格があります．コントロールプログラムと外部付加回路によってほとんどの方式に対応できる，きわめてぜいたくな機能を持ったLSIです．

このSIOのチップ（LSIの中のシリコンウェハ上の本体）は41個の外部端子を持っていますが，パッケージは40ピンであるため，外部ピンに接続されない端子があるもの2種類と，2本の端子を1本のピンに接続したものの合計3種類のSIOがあります．これをボンディングオプションと呼んでおり，SIO-0，SIO-1，SIO-2と区別しています．

SIOはきわめて多機能であるため，すべての動作を理解しようとせず，シリアル伝送の方式を理解したうえで，SIOの機能の必要な部分を使用するよう考えるのが効果的です．

●外部装置から送り込まれるデータバスの内容を，ある瞬間をとらえて（IO サイクルに）CPU 側のシステムデータバスに送り出す.
●CPU 側のシステムデータバスの内容を，ある瞬間をとらえて保持し，外部装置への受け渡しのタイミングをとる.

●外部装置から送られてくるデータ線上のパルス列を，8ビットの並列データになおし，CPU 側へ送り出す.
● 8ビットの CPU 側データバスの内容を，時間と共に順次，データ線へ送り出す.
●上記に必要なコントロール信号，内容の検査をするための付加情報を扱う.

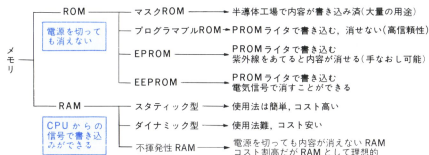
11 メモリの種類と用途

マイコンシステムで使われるメモリには何種類かありますが，現在ではほとんどが半導体素子で構成されるLSIメモリです．

LSIメモリは**ROM**（リードオンリーメモリ）と**RAM**（ランダムアクセスメモリ）に大別されます．ROMは書き込みに特別な装置が必要で，CPUからの書き込みはできません．ただし電源を切っても内容は消えないため，プログラムを入れておくのに主に使われます．RAMは，CPUからの書き込みができますが，電源を切ると内容は消えてしまいます．

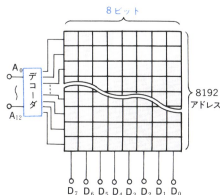
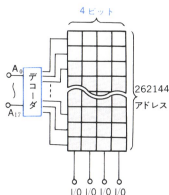
ROMには，**マスクROM**と**PROM**（プログラマブルROM）があります．マスクROMは半導体工場で生産する時点で内容が決定されてしまいますので，同一内容の量産に適しています．PROMは，ROMライタという装置で書き込みますが，一度書いたら変更できないもの（**バイポーラ型PROM**）と，電気信号で消去できる**EEPROM**（エレクトリカルイレーザブルPROM）と紫外線照射により消去できる**UVEPROM**（ウルトラバイオレットイレーザブルPROM）があります．一般的にはUVEPROMを単に**EPROM**と呼び，多く使用されています．

RAMは**SRAM**（スタティックRAM）と**DRAM**（ダイナミックRAM）があります．SRAMはフリップフロップにより構成されたメモリで，使いやすい特徴があります．DRAMはコンデンサの電荷蓄積を応用したメモリですから，時間と共に消えてしまいます．消えないうちに（実際には2ms以下ごと）に一度読み出して，再び同一内容を書き込むようにします．これを**リフレッシュ**といい，Z-80では，CPUが命令コードを解析している時間を利用して自動的にリフレッシュする機能を持っています．コスト的にはメリットがあるのですが，ドライブのインタフェースが複雑になり，むずかしさが残ります．最近ではメモリ内部にリフレッシュ機能を持ち，疑似的にSRAMと同様に使えるDRAMもあり，**疑似SRAM**と呼ばれています．また，電源を切っても内容の消えない不揮発性RAM，**NVRAM**（エヌブイラム）が普及しはじめ，理想的な半導体メモリとして，多方面に使われるようになってきています．

使い方による分類

主に

ROMはプログラムを書いておく　　読み出しだけだが電源を切っても消えないから

RAMは一時記憶用データ格納　　読み書きできるから

よく使われるメモリの例

メモリは容量，生産工程，パッケージなどが各種あります．メーカのマニュアルによって適当なものをさがすことも重要な仕事です．

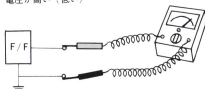
12

ビットパターンと16進表現

CPU内部や入出力信号線は，電気的に電圧が高い状態と低い状態の二つで，意味を持つようになっています．1本の電線では信号が有る，無い，の二つの状態でしかあり得ませんが，複数の電線を一束にして考えると，高・低の組合わせは2の累乗で増加します．アドレスバスは16本ありますので，16本の電線の高・低の組合わせは$2^{16}=65536$とおりになります．すなわち，CPUの出すアドレス情報は65536のメモリを識別できることになります．データバスは8本ですから，データとしては$2^8=256$とおりの種類が得られます．しかし数回に分割してやりとりをすれば，理論的には無限の組合わせが得られます．8本では数値としては0～255までしか表現できないはずですが，実際にはもっと広い範囲の数値を扱えるのはこのためです．

一つの2値状態を表現する単位を**ビット**と呼びます．アドレスバスは16ビットです．電圧の高い状態のことを“H”または“1”と呼び，低い状態のことを“L”または“0”と呼びます．16ビットの状態を表現するのに“1”と“0”を16個並べてもよいのですが，長くて扱いにくいので困ります．そこで“1”と“0”の組み合わせ（**ビットパターン**）を2進数とみなして，これを16進数に変換して扱うと大変便利になります．要するに，ビットパターンを4桁ずつ区切って“1”，“0”の組合わせに名前を付けたと考えればわかりやすいでしょう．名前は数字の0～9，次が10でもよいのですが，2桁になってしまうのでAと呼び，次にB，C～Fまであります．0～Fで16ありますので16進数になるのです．本書でもビットパターンをいちいち書くのは大変なので，16進数で表現します．その場合は後にHを付けて10進数と区別します．Z-80のアセンブリ語でもこのように書く決まりになっています．またマシン語の命令も本来はビットパターンで表現されるはずですが，一般的に16進数表現で扱います．

演算はビットパターンを2進数として行います．普通は0～255(8ビットでは)になりますが，符号を付けて－128～＋127として考えることもできます．どちらをとるかでプログラムは異なります．

❶　電圧が高い（低い）

❷　あながあいている（あいてない）

❸　スイッチが上に倒れている（下に倒れている）

❹　ランプが点燈している（消えている）

などを“1”というとき，（　）内のときは“0”という → 誰かが決めた

※　4つ要素が並ぶとすると下のとおり16とおりのパターン（組み合せ）が考えられる.

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

いちいち書くのは大変なのでパターンに名前をつけることにした（右欄）.

要素が4つだから ——→ $2^4=16$ とおり
要素が8つだったら →→ $2^8=256$ とおり
要素が16だったら ——→ $2^{16}=65536$ とおり

※　パターンを2進数と見なすと名前は16進数になる.
コンピュータ内の演算はこのパターンを2進数と見なすことにより数値を表現して行われるようになっている.

※　たった8本の電線（信号）でも256とおりの状態を表現することができる.

- “1”のことを“H”（high），“0”のことを“L”（low）と呼ぶこともある.
- 普通は“0”になっていて，“1”になったら「信号がある」と決めておけば，“1”のことをアクティブという.
- 普通は“1”になっていて，“0”になったら「信号がある」と決めておけば，“0”のことをアクティブといい「この信号は負論理だ」という.

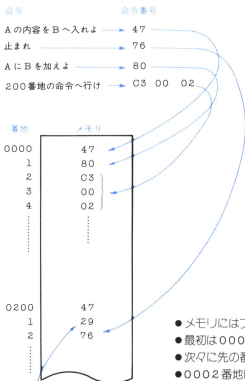
13 プログラムの実行

メモリに書き込まれているプログラムは，マシン語です．マシン語は英数字の組み合わせで，人間の目にはきわめてわかりにくいものですが，たとえば「AとBを加えてAに入れよ」とか，「止まれ」とか，「Aの内容を10番のポートへ出力せよ」といった命令にそれぞれ付けられた番号なのです．CPUは，日本語や英語で書かれるより，番号のほうが簡単に見分けることができます．

プログラムの書き込まれたメモリには，これも番号＝番地（**アドレス**）が付けられています．実行するときは，まずゼロ番地に書かれた命令から順次1ずつくりあげていきます．ところが命令の中に「何番地へ飛べ」というのがあると，1ずつくりあげるのではなく，指定の番地へ飛び（ジャンプ）ます．また「前の演算の結果がゼロならば何番地へ飛べ」という命令では，条件によって，ゼロなら飛び，ゼロでなければ次の番地の命令を実行します．

サブルーチンコール命令があると，指定された番地へ飛び，そこから順次実行し，最後に付けられたリターン命令で，先ほど飛んできた番地の次へ戻ることができます．同じ手順を何度も使いたいときによく使う方法で，飛ぶ前のプログラムを**メインルーチン**，飛んでくるプログラムを**サブルーチン**と呼びます．

サブルーチンとよく似ているのが割り込みです．**割り込み**は，命令があって特定の番地へ飛ぶのではなく，電気信号がCPUの割り込み信号端子に与えられると，どこの命令を実行中であってもある特定の番地へ飛び，リターン命令で元のメインルーチンへ戻ります．この特定の番地に置かれたプログラムを**割り込み処理ルーチン**と呼ばれます．

CPU内には，プログラムカウンタ(PC)と呼ばれるレジスタがあり，この内容で，いま実行すべき命令の入っているメモリのアドレスを示すようになっています．一つの命令を実行するたびにこのPCを増やし，次の命令のアドレスを指します．また，特定のアドレスへジャンプするようなときは，PCへ飛び先のアドレスを強制的に入れることにより目的を達成します．

（注） ここではビットパターンの名前で書いたが，メモリにはビットパターンそのものが記憶されている．

- メモリにはプログラムを命令番号で入れておく．
- 最初は0000番地の命令を実行する．
- 次々に先の番地を順序よく実行する．
- 0002番地にある命令（200番地へ行け）を実行すると，次は200番地にある命令を実行する．
- 次々に先の番地を順序よく実行する．
- 0202番地にある命令（止まれ）を実行すると，そこで以後の実行をやめ停止する．

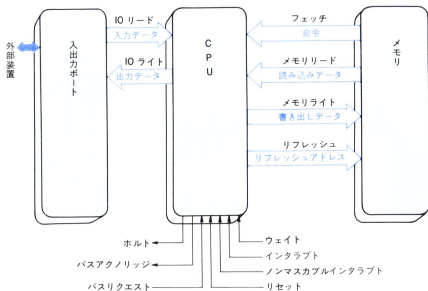
14

CPU の信号のやりとり

CPU にはたくさんの信号入出力線が出ていますが，これらの信号線を使って，基本的な六つの動作を時分割的に行います．それぞれを**マシンサイクル**と呼びます．

1. **フェッチサイクル**（M1 サイクル）
   メモリに書き込まれているプログラム命令を CPU 内の命令解析用レジスタ（インストラクションレジスタ）へ読み込み，解読する．
2. **メモリリードサイクル**
   メモリからデータを CPU 内のレジスタへ読み込む．
3. **メモリライトサイクル**
   CPU 内のレジスタからデータをメモリへ書き出す．
4. **IO リードサイクル**
   入力ポートからデータを CPU 内のレジスタへ読み込む．
5. **IO ライトサイクル**
   CPU 内のレジスタからデータを出力ポートへ書き出す．
6. **リフレッシュサイクル**
   DRAM をリフレッシュするためのアドレス信号を出す．

以上のほかに，一連のサイクル動作ではありませんが，単独の意味を持つ信号が七つあります．

1. **ウエイト**；クロックサイクル数を増やす．
2. **インタラプト**；割り込み要求．
3. **ノンマスカブルインタラプト**；**ノンマスカブル割り込み要求．**
4. **リセット**；初期状態に戻す．
5. **バスリクエスト**；CPU の動作を停止させバスを空け渡す要求．
6. **バスアクノリッジ**；バスリクエストを受け付けた返事．
7. **ホルト**；CPU がホルト命令を実行し，停止状態に入ったことを外部へ知らせる．

⇨ は一連の動作（マシンサイクル）で，アドレス情報とデータの受け渡しを伴う．

──► は情報の受け渡しは伴わない．

15 データバス，アドレスバスとシステム制御信号

CPUにある8本の**データバス**は，CPUと外部とのデータを出し入れするための信号線です．CPUの動作サイクルによって乗ってくるデータの意味は異なります．フェッチサイクルでは，メモリから命令コードが乗ってきますし，メモリリードサイクルではメモリからデータが乗ってきます．またメモリライトサイクルとIOライトサイクルでは，CPU内のレジスタからのデータがデータバス上に乗せられます．

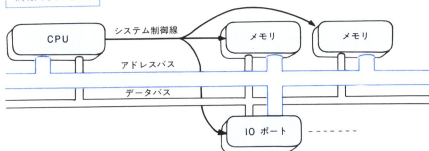
このようにデータバスは，その時々によって種々の信号の出し入れに使われます．

また，**アドレスバス**もフェッチサイクルでは実行すべき命令の入っているアドレスが出力され，メモリリード，ライトサイクルでは，読んだり書いたりするメモリのアドレスが出力され，また，IOライト，IOリードサイクルでは，入出力すべきIOポートのアドレスが乗せられてきます．

データバスにせよ，アドレスバスにせよ，目的の異なる信号が次々に乗せられますので，現在の信号は何を意味するかを識別するための，別の信号線が必要になります．リード($\overline{\mathrm{RD}}$)，ライト($\overline{\mathrm{WR}}$)，メモリリクエスト($\overline{\mathrm{MREQ}}$)，IOリクエスト($\overline{\mathrm{IORQ}}$)，エムワン($\overline{\mathrm{M1}}$)，リフレッシュ($\overline{\mathrm{RFSH}}$)の各信号の組み合わせが，この役割をしています．実際にはこれらの各信号がすべて同時に出たり，止まったりするのではなく，それぞれのタイミングで変化します．アドレスバス，データバスの占有時間も目的により異なります．

CPUの信号のうちデータバスとアドレスバスだけは正論理，すなわち，1のとき“H”レベル，0のとき“L”レベルとなります．他の信号線はすべて負論理で，普段信号のないとき“H”になっていて，必要なときだけ“L”になります．

負論理の信号名の略称には——（バー）を付けて表わします．

情報の受け渡し

システム制御信号線

エムワン（$\overline{M1}$） メモリリクエスト（$\overline{MREQ}$） IO リクエスト（$\overline{IORQ}$） リード（$\overline{RD}$） ライト（$\overline{WR}$） リフレッシュ（$\overline{RFSH}$）	これらの組合せでアドレスバスに乗っている情報，データバスに乗っている情報あるいは，CPU がデータバスにどこから情報を乗せてもらいたいかを表現している．

※CPU は命令の内容によって，送り出すべき情報を送り出し，要求すべき情報を受けとる．
情報はデータバスを使う．
送り先や要求先は，システム制御信号とアドレスバスで指定する．

※メモリや IO ポートなど周辺回路は，上の CPU の要求に対しある時間内に正確に応答しなければならない．これは汎用 CPU の場合ユーザの利用技術にかかっている．

16 命令語の構成

マシン語の命令は，一つの機能を持つ命令が1～4バイトで構成されます．

1バイト命令は，オペコードだけで意味を持つ命令です．

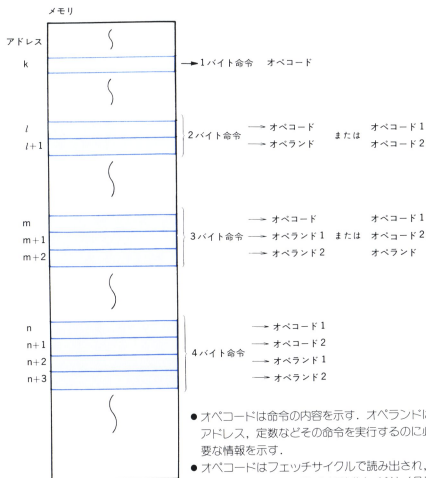
2バイト命令は，オペコードと，オペランドに記述された数値を次の1バイトに持つものと，2バイトで一つのオペコードを意味するものがあります．

3バイト命令は，オペコードと，オペランドに記述された数値を次の2バイトに持つものと，2バイトのオペコードと1バイトのオペランドを持つものがあります．

4バイト命令は，2バイトのオペコードと2バイトのオペランド情報により構成されます．

オペコードが2バイトにわたる命令はZ-80特有のものです．最初の1バイトをフェッチし解読した時点で，2バイトのオペコードを持つことがわかりますから，あと1バイトをフェッチし解読します．したがって，フェッチサイクルが二つあり，エムワン（$\overline{\text{M1}}$）信号も2回出されます．

CPUは，この1～4バイトで構成される命令をフェッチサイクルとメモリリードサイクルを起動して全部読み込み，意味を解析し実行します．これが終わると次の命令へと順次読み込み，解析，実行をくり返していきます．

一つの命令を実行する一巡を，**命令サイクル**（インストラクションサイクル）と呼びます．

- オペコードは命令の内容を示す．オペランドはアドレス，定数などその命令を実行するのに必要な情報を示す．
- オペコードはフェッチサイクルで読み出され，オペランドはこれに続くメモリリードサイクルで読み出される．

17

命令の実行

例として〔LD　A,（lm）〕という命令を分解してみましょう．

この命令には，3Aという番号が付いていて，CPU内部では，この番号で意味がわかるようにできています．3Aの次に二つの情報，lとmが連続している3バイト構成の命令です．具体的には「lm番地のメモリの内容をCPU内のAレジスタへ入れよ」という意味があります．lとmは各8ビットで，メモリアドレスは16ビットで表わされますので，二つで一つのアドレス情報になります．このときメモリ上にはl（上位）とm（下位）を反対にして，m，lの順で並べる規定になっています．

M1サイクル（フェッチサイクル）

プログラムカウンタ（PC）の内容で示される番地から3Aをフェッチする．CPU内のインストラクションレジスタへ入れてこの意味を解析すると，次に何をすべきかわかり以下を実行する．PCに1を加える．

M2サイクル（メモリリードサイクル）

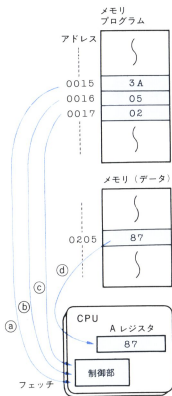
PCの内容番地よりmを読み込んで制御部のレジスタへ一時格納する．PCに1を加える．

M3サイクル（メモリリードサイクル）

PCの内容番地よりlを読み込んで制御部のレジスタへ一時格納する．PCに1を加える．

M4サイクル（メモリリードサイクル）

制御部のレジスタへ格納したlとmにより一つのアドレスlmを構成し，lm番地の内容を読んでAレジスタへ入れる．

以上の四つのマシンサイクルによって命令を実行したわけですが，命令の内容によって，それぞれのサイクルの組み合わせは異なり，サイクル数も異なります．ただし，どんな命令でも必ず，フェッチサイクルが最初に起動され，フェッチした命令の内容によって次に何をするか決定します．

命令実行の一例

① 0015番地の命令を実行する順番がくると，CPUはアドレスバスに0015を乗せ，システム制御信号はメモリからの命令読み込みを要求する．
② メモリはこれに対し0015番地の中身をデータバスに乗せる．
③ CPUはこれを受けとり命令の意味を解析する．

以下解析の結果

④ CPUはアドレスバスに0016を乗せ，システム制御信号によりメモリからの読み込みを要求する．
⑤ メモリはこれに対し0016番地の中身をデータバスに乗せる．
⑥ CPUはこれを受けとる．
⑦ 同様に0017番地の中身も受けとる．
⑧ いま読み込んだ0016と0017番地の中身をつないで0205という値を作り，これをアドレスバスに乗せ，システム制御信号によりメモリからの読み込みを要求する．
⑨ メモリはこれに対し0205番地の中身をデータバスに乗せる．
⑩ CPUはこれを受けとりAレジスタへ入れる．

（①～③がフェッチサイクルで，各命令実行ごとにまずこのサイクルに入る．）

18

アセンブラ記法のルール　-1-

16進数で表現されるマシン語命令では，人間には覚えにくくわかりにくいため，直感に訴えるような英語の略語（ニモニック）を対応づけたのが**アセンブリ言語**です．アセンブリ言語で書かれたプログラムを**ソースプログラム**と呼びます．ソースプログラムをマシン語に，自動的に置き換えるプログラムが**アセンブラ**です．このアセンブラプログラムが読み込むソースプログラムの書き方には，一定のルールがあります．

ラベルはアドレスに付ける仮の名称です．この命令のアドレスへジャンプしたり，参照したりするために必要なところだけでよく，付けなくてもよいのです．英文字で始まる6文字以内の英数字列で，後で見てわかりやすい名前を付けます．

オペコードは命令語のニモニックです．CPUの持つ命令の中から必要なものを選んで使います．

オペランドは，オペコードによってはないものもあります．一つだけのものも，“，”で区切って二つ必要なものもあります．二つ必要な命令の場合は，前に書くのが**ディスティネーション**，後に書くのを**ソース**といって，ソースからディスティネーションへのデータの移動を意味します．ソースまたはディスティネーションをかっこでくくるときは，かっこ内のアドレスのメモリに入っている内容を意味します．かっこ内がレジスタ名のときは，そのレジスタに入っている内容をアドレスとしたメモリを意味します．

コメントはプログラムをあとで見たときに，わかりやすくするために必要なことをメモしておく所です．的確なコメントを豊富に付けられたプログラムは，誰が見てもよくわかり，大変便利なものです．

アセンブリ言語で書く命令には，上のようにマシン語命令に変換されるもののほかに，疑似命令といって，マシン語に変換されずに，アセンブラプログラムに対して指示を与えるだけの命令もあります．また，アセンブラプログラムが**マクロアセンブラ**と呼ばれる場合は，1命令をあらかじめ別に定義された数個以上のマシン語命令群に変換するマクロ命令といったものも使うことができます．

オペランドの表記法

	ディスティネーション	ソース
レジスタ名	レジスタ	レジスタ
定数	――――	定数
(レジスタ名)	レジスタの内容アドレスのメモリ	レジスタの内容アドレスのメモリ
(定数)	定数アドレスのメモリ	定数アドレスのメモリ

16進表現で，最上位がA～Fのアルファベットになるときは頭に0を付け，ラベル名と区別する．
(例)　A000HはOA000Hと書く

19 アセンブラ記法のルール -2-

行の先頭，すなわち前の行の最後に付けられたC/R（キャリッジリターン）の次に続く文字列は，**ラベル**です．英大文字に続く英数字の組み合わせで，6文字以上は無視されます．ラベルの文字列の直後に“：”コロンがあれば，行頭でなくてもラベルとみなされます．ラベルを付けない行は，1個以上のスペースをラベル代わりに入れておきます．

ラベルと1個以上のスペースで区切られた後にオペコードを書きます．命令表の中から選んでください．全部で74あります．

オペコードと1個以上のスペースで区切られた後には，そのオペコードに対するオペランドが続きます．二つ必要なときは“，”コンマで区切ります．このオペランドは，レジスタ名，16進表現の定数，10進表現の定数，フラグの状態(分岐の条件）名，を書きます．定数は，他の命令に付けたラベルを書いてもよく，ジャンプする場合などは特に絶対番地を計算しなくてもよいので便利です．オペランドのソースやディスティネーションに（　）かっこを付けたときは，その内容を番地とするメモリの内容を意味します．

オペランドに書く定数は，10進表現で書いてもよいし，16進表現で書いてもよいのですが，16進表現で書くときは，後にH（ヘキサデシマルの頭文字）を付けておきます．また，先頭がアルファベットA～Fで始まるときはさらに前に0を付けてラベルでないことを特徴づける約束になっています．

コメントは行中のどこにでも“；”セミコロンがあれば，それ以後は自由にコメントエリアとして使うことができます．ただし1行は通常80字位までで切られることがありますので，長くなる場合は数行に分けます．

Z-80のアセンブリ言語は標準的な取り決めがありますが，アセンブラプログラムの種類（メーカや機種）により若干の違いがあります．アセンブラを使うときは，それぞれの説明書を一読してください．

コメント

```
                Z80 MACRO ASSEMBLER  V1.1  PAGE    1

 1                       ;IPL
 2                               ORG*     0FF00H
 3 FF00 3EED                     LD       A,0EDH  ;MODE
 4 FF02 D3F3                     OUT      (0F3H),A        ;MDR0
 5 FF04 3E37                     LD       A,037H  ;SLP
 6 FF06 D3F2                     OUT      (0F2H),A        ;CDR
 7 FF08 DBF6             LP0:    IN       A,(0F6H)        ;ISR
 8 FF0A CB7F                     BIT      7,A     ;CCE
 9 FF0C 20FA                     JR       NZ,LP0  ;CCE WAIT
10 FF0E 0610                     LD       B,16    ;BLOCK COUNT
11 FF10 210000                   LD       HL,0000H        ;ADDRESS
12 FF13 3EFF             LP1:    LD       A,0FFH  ;WDC SET
13 FF15 D3F1                     OUT      (0F1H),A        ;WDC
14 FF17 3E3B                     LD       A,03BH  ;RDL
15 FF19 D3F2                     OUT      (0F2H),A        ;CDR
16 FF1B DBF6             LP2:    IN       A,(0F6H)        ;ISR
17 FF1D CB6F                     BIT      5,A     ;DA
18 FF1F 2007                     JR       NZ,LP3
19 FF21 DBF0                     IN       A,(0F0H)        ;DBR
20 FF23 2F                       CPL
21 FF24 77                       LD       (HL),A  ;DATA LOAD
22 FF25 23                       INC      HL      ;ADDRESS COUNT UP
23 FF26 18F3                     JR       LP2
24 FF28 CB7F             LP3:    BIT      7,A     ;CCE
25 FF2A 20EF                     JR       NZ,LP2  ;CCE/DA  WAIT
26 FF2C 10E5                     DJNZ     LP1     ;END?
27 FF2E C300E0                   JP       EXIT    ;RETURN TO MONITOR
28                       ;
29      E000             EXIT:   EQU*     0E000H
30                               END*
```

定数（番地）の代りにラベルで指定

コメント行（見やすくするために一行あける）

* ORG：以下のプログラムを配置するアドレスを指示する．
EQU：ラベル名の文字列をオペランドの数値とすることを指示する．
END：プログラムの終りを明示する．

20

命令の分類

機能別に命令を分類すると，次のようになります．

8 ビットロード

1バイトの情報をレジスタ間，レジスタとメモリ間で転送する．1バイト定数をレジスタかメモリへ入れる．

16 ビットロード

2バイトの情報をレジスタ間，レジスタとメモリ間で転送する．2バイト定数をレジスタへ入れる．

レジスタ交換

レジスタの内容を入れ替える．

メモリブロック転送

メモリ内の複数バイトのブロックを別のアドレスへ転送する．

メモリブロックサーチ

メモリ内ブロックに指定の情報があるかどうか探す．

8 ビット演算，論理演算

1バイト単位の加減算，論理和，論理積，排他的論理和，カウントアップ，カウントダウン，比較をする．1または2の補数をとる．

16 ビット演算

2バイト単位の加減算，カウントアップ，カウントダウンをする．

ローテートシフト

レジスタ，メモリのビットパターンを回転させ，または左右へずらす．

10 進補正

2進化10進数として扱うデータの加減算後の補正をする．

フラグ操作

キャリフラグを“1”にする．反転させる．

CPU 制御

プログラムの実行と，割り込みを制御する．

ビ ッ ト 操 作

レジスタ，メモリの特定の1ビットを“0”か“1”にする．また，判定する．

ジ ャ ン プ

プログラムの実行を条件によりまたは無条件で，指定のアドレスへ移行させる．

コール，リターン，リスタート

プログラムの実行を条件によりまたは無条件で，指定のアドレスへ移行させる．ただしこのとき，リターン命令によりもとのアドレスへ戻れるような手順を含んでいる．

入 力 出 力

指定のIOポートへレジスタとの間で1バイト情報を入・出力する．

連 続 入 出 力

メモリブロックを1バイト単位に連続して出力する．

メモリブロックへ1バイト単位に連続して入力する．

マイナス数の表現（2の補数）

2進数でマイナスを表現するには，多くの場合次の手法がとられます．

4ビットだけで考えれば，ゼロすなわち0000のひとつ前の－1は1111になります．なんとなればこれに1をたすと桁上りをして10000ですが，4ビットだけで考えているので桁上りを無視せざるを得ないからです．

上記のとおり，4ビットでは0～F（10）または－8～＋7を表現することができます．正数を負数に変えるには，ビットの“1”，“0”を反転させて1を加えればよいのです．

21 メモリ空間とIO空間

Z-80では接続できるメモリは最大64Kバイト（Kは1024）です．これはアドレスバスが16本あるので，$2^{16}=64$Kとなるからにほかなりません．メモリとは別にIOの領域として256のアドレスがあります．したがって，IOポートは256まで接続できることになります．これは，IOリード，IOライトサイクルではアドレスバスの下位8ビットのみ使用しているためで，$2^8=256$ということです．同じアドレスバスをメモリリクエスト信号とIOリクエスト信号により切り換えて，使用するためにメモリ空間とIO空間を別々に持ったことになります．

メモリ空間へのアクセスは，ロード[LD]命令を使い，IO空間へのアクセスは，イン［IN］，アウト［OUT］命令を使います．IO空間へのアクセスのほうが，タイミング的に長い時間がとれるようになっています．

アセンブラ記法でIO空間のアドレスを示す場合はOUT(OOH)，Aのようにかっこに入れる決まりになっています．

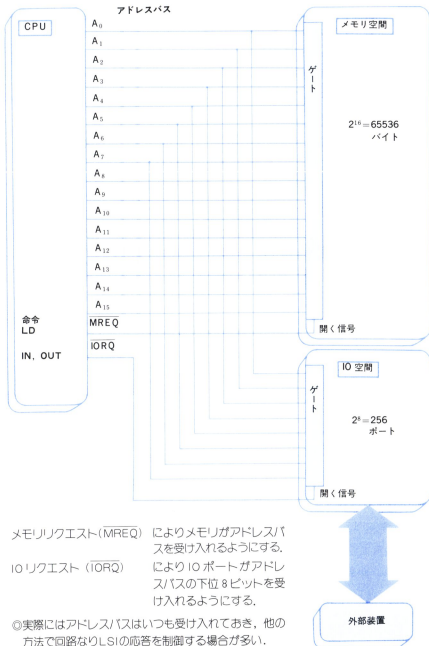
バイト　○8ビットを1バイトと呼ぶ．8ビットのCPUはバイトマシンと呼ばれ，8ビットを同時に扱うので，メモリは1アドレスのアクセスで8ビット（データバス8本）の読み書きが行える構成とする．

○したがって，1アドレス4ビットのメモリ（2114など）では2個，1アドレス1ビットのメモリ（4116など）では8個並列に並べる．

メモリリクエスト（$\overline{MREQ}$） によりメモリがアドレスバスを受け入れるようにする。

IOリクエスト（$\overline{IORQ}$） によりIOポートがアドレスバスの下位8ビットを受け入れるようにする。

◎実際にはアドレスバスはいつも受け入れておき，他の方法で回路なりLSIの応答を制御する場合が多い。

22

アドレスデコーダ

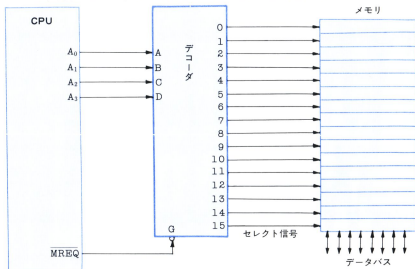
アドレスバスに出力されるビットパターンは，16本の端子から65536とおりもありますから，どのパターンのときはどのメモリを選ぶかを決めてやらなければなりません．

アドレスバスに出てくるアドレス信号のビットパターンから特定のメモリやIOポートを選び出すセレクト信号（イネーブル信号）を作り出すわけです．メモリはたとえば，256バイトのメモリであれば，LSIの中に256バイトすなわち，アドレスバス8本分のデコーダを持っていて，内部で特定のメモリセルを選択してくれます．ところが，このメモリを複数使用するときや，IOポート用のペリフェラルを複数使用するときは，CPUとペリフェラルやメモリの間にアドレスデコーダが必要になります．デコーダは通常74 HCシリーズなどの標準ロジックICで構成します．

番地だけでなく，メモリ空間とIO空間の切り換えも，ここで行なうのが普通です．メモリリクエスト（$\overline{\mathrm{MREQ}}$）がアクティブつまり，負論理出力ですから“L”になったときはメモリがイネーブルに，IOリクエスト（$\overline{\mathrm{IORQ}}$）がアクティブならばIOポートがイネーブルになるようなロジックを組むのです．通常問題になることは少ないのですが，メモリやIOポートへセレクト信号を出してから実際にイネーブルになり，データを受け入れまたは出力するまでの時間は，CPUの動作時間に適合しなければなりません．CPUより遅くて間に合わない場合は，ウエイト信号をCPUに与えて一時待たせるなどの方法をとります．メモリやIOポートの応答時間と，アドレスデコーダなどの遅れ時間を含めて考える必要があります．しかし一般には，この遅れは無視できる場合のほうが多いようです．

アドレスデコーダとしてよく使われる標準ロジックICは74 HC 139や74 HC 155などがあります．規格表から目的に合うものを探し出し，価格や入手状況を検討して決定してください．

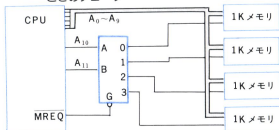
例 16バイトメモリのアドレスデコーダ

16バイト$=2^4$：4本のアドレスバスで16のアドレスを表現できる

デコーダの出力

（X は何でもよい）

G	D	C	B	A	出力
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	×	×	×	×	ナシ

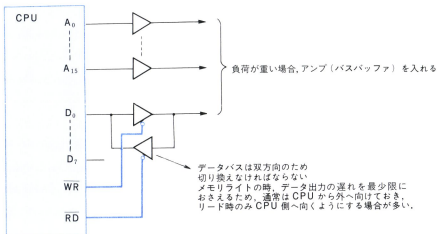
- $\overline{\text{MREQ}}$ がノンアクティブ（負論理だから“1”）のときはメモリセレクトはしない．
- $\overline{\text{MREQ}}$ がアクティブ（“0”）のときだけ A_0 から A_3 のビットパターン16種により16本の出力のどれかへセレクト信号を出す．
- セレクトされたメモリはデータバスとデータの受け渡しを行う．

参考 1Kバイトのメモリ4個を使ったときのデコーダ　［$1K=1024=2^{10}$］

23 バスバッファ

TTLのロジックICには，ファンイン，ファンアウトという概念があります．マイコンの周辺の場合“H”と“L”のほかに，電気的に切り離されたハイインピーダンスの3種類の状態をとるスリーステートの端子が多いため，単純には考えられなくなります．Z-80ファミリの出力端子は，一つのノーマルタイプのTTLをドライブできる能力を持っています．LSタイプであれば四つまで可能です．これ以上の素子をドライブしなければならないときや，接続ケーブルを長く引き回すときは，バッファを入れなければなりません．特に長いケーブルを使用するときは波形の乱れやノイズ防止のため，バスドライバ，レシーバの能力に特別の考慮が望まれます．ときにはフォトカプラやシュミットトリガが使われることもあります．

メモリなどが何枚ものボードで構成される場合，ボードの出入口のところにバスバッファを設けるのが普通です．単にバッファだけでなく，ゲートの働きもして，そのボード内のアドレスがアクセスされたときだけゲートが開くようにしておけば，CPUに対して無駄な負荷をかけることもなく，デバッグもやりやすくなります．

データバスに使用するバッファは双方向性です．リード，ライトの信号によって方向を決定します．マルチCPUシステムとしたときや，DMAを使用したときはきわめて複雑になりますので，システムのブロック化をするときには十分考慮する必要があります．

割り込みを使用するシステムでは，ペリフェラルは，CPUがメモリからフェッチしてくる命令を横からデータバスを見ていて，割り込みからの復帰を知るようになっています．したがって，データバスのバッファはフェッチサイクルではメモリからCPUへ向くと同時に，メモリからペリフェラルへも送り込まなければなりません．

●ゲートはユニットの機能により, 開け閉めする場合もあるが, 常時開けておき, 他のゲートを制御するための情報を受けとる場合もある.

24

システムクロック

Z-80ファミリは，システムのすべての動作はシステムクロックに同期して進められますので，CPUやペリフェラルのクロック（ϕ）端子へ共通のクロックパルスを与えなければなりません．当然周波数が高いほど処理速度は速いわけですがLSIの性能で上限は規定されます．また周辺に接続されるメモリや入出力の端末の応答速度が追従しなければ，なんの意味もなく，単にCPUを待たせる回路が増えるだけになります．

システムクロック（ϕ）は単相の方形波ですが，“H”と“L”の幅がそれぞれ規定されています．したがって，周波数が規定以下であっても上限に近いときはデューティ比が50%に近くないと具合がわるい場合があります．そのときは目的のクロック周波数の2倍の周波数で発振させ，1/2分周すれば正確にデューティ比50%のパルスが得られます．

クロックパルス1周期を**クロックサイクル**と呼びます．マシンサイクルは，クロックサイクル数個によって一つずつ進められます．1クロックサイクルは，クロック周波数の逆数〔秒〕ですから，たとえばクロック周波数が4MHzなら$1/(4\times10^6)=0.25\mu s$，8MHzなら$1/(8\times10^6)=0.125\mu s$となります．

TTL，7404による簡単な発振回路

限度いっぱいの周波数で働かせるとき

限度いっぱいの周波数で働かせるとき

理想的な回路例

（ザイログ社マニュアルより）

25

リセット

CPUのリセット($\overline{\text{RESET}}$)端子へのリセット信号は，パワーオン時には与えなければなりません．CPU以外にも同じ信号を与えます．**PIO**と**DMA**はパワーオンリセットの機能を内部に持っていますので必要ありませんが，与えてもよいのです．リセットの時間は，電源電圧が安定し，クロックが安定に与えられてから最低3クロックサイクル分（2.5MHzクロックであれば，$1/(2.5\times10^{6})=0.4\mu s$．したがって，$0.4\times3=1.2\mu s$）以上です．実用上問題なければ，長いほど安全といえます．

CPUは，このリセット信号により

プログラムカウンタ（PC）

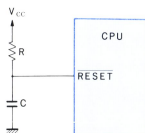
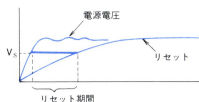
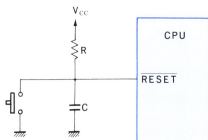
Iレジスタ

Rレジスタ

をゼロにして，割り込みモードを0として，割り込みを受け付けない（インタラプトディスエーブル）状態にします．これ以外のレジスタ（もちろん，CPU外のRAMも）は変化しません．

リセット信号がノンアクティブ（負論理ですから“H”）になると，まず，ゼロ番地の命令をフェッチすることから動作を始めます．もしIレジスタを使用するなら，ゼロ番地からのプログラムでIレジスタへ必要な数値を書き，割り込みモードを希望のモードに設定し，その他必要な初期化を行なってから，割り込みを受け付け可の状態（インタラプトイネーブル）にする〔EI〕命令を実行させます．またスタックポインタ（SP）の設定もしなければなりません．

リセット信号は，パワーオン時に与えることはもちろんですが，プログラムが意図しない無限ループに入ってしまうこともあり，特にデバック中には，手動で電源をオンオフせずにリセットできるようにしておくと大変便利です．ただし，なんらかの動作中に不用意にリセットすると困ることもありますので注意が必要です．リセット期間中はメモリリフレッシュ信号は出ません．

パワーオンリセット

R と C により数 ms の時定数を持たせる

マニアルリセット

パワーオンリセットも兼ねる．スイッチのチャタリングは
C により吸収される．

＊ リセット信号にノイズがのることを想定すると，シュミットトリガ回路を通した方が確実といえる．

26

システム構成

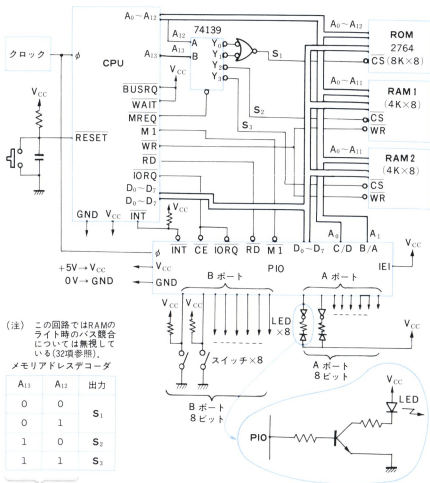
マイコンシステムの構成は，目的によって大きく異なります．普通はCPU 1個に対して，入出力のポート数あるいはビット数を満たすだけのIOペリフェラル，そしてプログラムを収容しきれるだけのROMと，データエリア（ワーキングエリア）としてのRAM，IOとメモリのアドレスデコーダ，クロック，リセットで構成されます．さらに大きなシステムではDMAを採用したりCTCを追加したり，場合によってはCPUを複数にしたマルチCPUシステムとすることもあります．マルチCPUシステムは，目的の仕事がはっきり区分できるときはプログラムが楽になり，効率がよくなる点でメリットがあります．

図は入力にスイッチ8個，出力にLED 8個，プログラムは8Kバイト以下，データエリアも8Kバイト以下，といった簡易なコントローラの例を示したものです．メモリは合計16Kバイトですので，アドレスバスは下位14ビットだけを使い，上位2ビットはあそばせてあります．したがって，14ビットだけが有効で上位桁は何であってもよいわけです．何であってもよいということは，何であってもいずれかのメモリがセレクトされてしまうので，これ以上増設することはできなくなります．IOポートにはPIOを1個使ってありますので，アドレスは四つだけ必要（詳細は**PIO**の項参照）になり，下位2ビットだけ有効です．

このシステムでもスイッチをマイクロスイッチや光センサに，LEDをリレーやソレノイドに変えれば，メカニズムのコントロールロジックを置き換えたり，簡単なシーケンスコントローラとして実用化することができます．

（注） この回路ではRAMのライト時のバス競合については無視している（32項参照）．

メモリアドレスデコーダ

A_{13}	A_{12}	出力
0	0	S_1
0	1	S_1
1	0	S_2
1	1	S_3

メモリアドレス配置

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	16進表現
ROM	×	×	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	×	×	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
RAM1	×	×	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	×	×	1	0	1	0	1	1	1	1	1	1	1	1	1	1	2FFF
RAM2	×	×	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000
	×	×	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF

各メモリLSIの中でデコードしてくれるので最小値と最大値のみを記入した

27 フェッチサイクルの動作

一つの命令を実行するとき，最初にCPUはフェッチサイクルに入ります．プログラムカウンタの値に示されるアドレスのメモリからオペコードを読み込み，解析します．そしてプログラムカウンタに1を加え，次の読み込みに備えます．

フェッチサイクルが始まるとアドレスバスにプログラムカウンタの値が乗せられ，同時にエムワン（$\overline{\mathrm{M1}}$）信号が“L”すなわちアクティブになります．アドレスバス上のデータの安定するのを待ってメモリリクエスト（$\overline{\mathrm{MREQ}}$）信号が，続いてリード（$\overline{\mathrm{RD}}$）信号がアクティブになってメモリをアクセスします．3番目のクロックT_3の立ち上がりで，CPUはデータバスの内容を読み込み（サンプリング）します．T_2が立ち下がるときにウエイト（$\overline{\mathrm{WAIT}}$）信号がアクティブになっていると，そのままの状態でウエイトサイクルT_WがT_2とT_3の間に入ります．T_Wの立ち下がりでもウエイト（$\overline{\mathrm{WAIT}}$）信号を調べますので，次へ進めたいときはウエイト（$\overline{\mathrm{WAIT}}$）を，ノンアクティブ“H”に戻さなければなりません．応答の遅いメモリを使うときに，これで時間待ちさせることができます．T_3ではDRAMをリフレッシュするためのリフレッシュアドレスがアドレスバスの下位7ビットに出され，リフレッシュ（$\overline{\mathrm{RFSH}}$）信号も同時にアクティブになります．メモリリクエスト（$\overline{\mathrm{MREQ}}$）信号はアドレスバスが安定しているであろう期間にアクティブになります．

リフレッシュ中は，CPUの内部では命令の解析が行われ，次のサイクルに何をすべきか判断していきます．8ビットの汎用レジスタ間の転送命令などでは4クロックのフェッチサイクルだけで実行を完了してしまいます．

フェッチサイクルのメモリリクエスト（$\overline{\mathrm{MREQ}}$）信号は，データバスをサンプリングする約1.5クロック前に出されますので，この時間内にメモリが応答しなければなりません．しかし問題になることは少なく，万一のときもウエイト（$\overline{\mathrm{WAIT}}$）信号で解決できます．

クロックサイクル　マシンサイクル

フェッチサイクル

リフレッシュサイクル

T_1　T_2　T_3　T_4　4クロックサイクル必要

クロック

アドレスバス　プログラムカウンタ　Rレジスタ

$\overline{MREQ}$

$\overline{RD}$

$\overline{M1}$

$\overline{RFSH}$

データバス

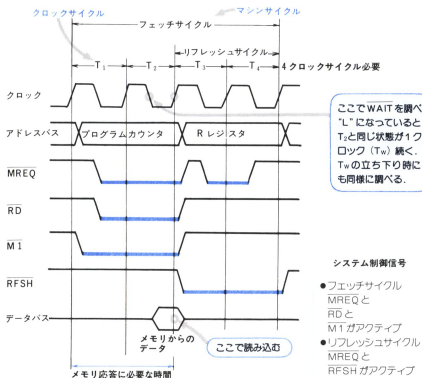
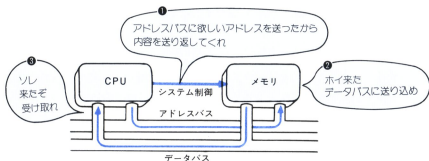
メモリからのデータ

ここで読み込む

メモリ応答に必要な時間

ここで$\overline{WAIT}$を調べ“L”になっているとT_2と同じ状態が1クロック（T_W）続く．T_Wの立ち下り時にも同様に調べる．

システム制御信号

- フェッチサイクル
  $\overline{MREQ}$と
  $\overline{RD}$と
  $\overline{M1}$がアクティブ
- リフレッシュサイクル
  $\overline{MREQ}$と
  $\overline{RFSH}$がアクティブ

28 メモリリードサイクル

メモリからデータを読み出して，レジスタや他のメモリへ転送する命令の1サイクルとして，メモリリードサイクルへ入る場合と，命令語のオペランドを読み出すためのメモリリードサイクルがあります．動作はどちらも同じで，通常*1 3クロックサイクルの間に実行されます．

アドレスバスに読み込むべきアドレスが送出され，これが安定するのを待ってメモリリクエスト（$\overline{\text{MREQ}}$）信号とリード（$\overline{\text{RD}}$）信号が同時に出されます．2番目のクロックは，メモリが応答し，データバスへ情報を乗せてくるのを待つ時間です．もしメモリの応答が間に合わない場合には，立ち下がりの前後にウエイト信号を入れることができます．3番目のクロックの立ち下がり時点に，CPUはデータバスのゲートを開き，内部へデータを読み込みます．

リード（$\overline{\text{RD}}$）信号とメモリリクエスト（$\overline{\text{MREQ}}$）信号が出てからCPUが読み込むまでの時間は，フェッチサイクルでは1.5クロックでしたが，メモリリードサイクルでは2クロックあります．クロックを2.5MHzとすると，1クロック当りの時間は400ns*2 ですから，フェッチサイクルのときは600ns，メモリリードサイクルのときは800nsとなります．したがって，アクセスタイムが450ns以下のメモリであれば問題なく使用できます．

＊1　命令により例外はある．
＊2　ns：ナノセックといい，10^{-9}秒．

マシンサイクル

メモリリードサイクル

クロックサイクル

T_1　T_2　T_3

クロック

アドレスバス　メモリアドレス

$\overline{\text{MREQ}}$

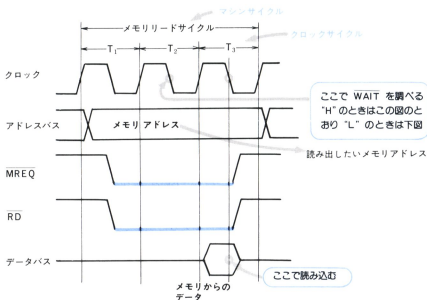
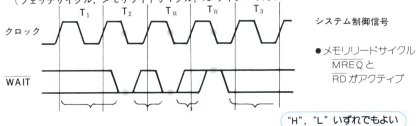
$\overline{\text{RD}}$

データバス

メモリからのデータ

ここで WAIT を調べる
"H" のときはこの図のとおり "L" のときは下図

読み出したいメモリアドレス

ここで読み込む

T_W では信号は T_2 と同じ状態を保っている

29 メモリライトサイクル

CPU からメモリに対する書き出しのマシンサイクルです．レジスタからメモリ，メモリからメモリへの転送命令などで実行されますが，サブルーチンコールのコール命令で，戻り番地をスタッカへ格納するときや，割り込みがかかったときも実行されます．いずれも同じタイミングで通常* 3 クロックです．前項のメモリリードサイクルとほとんど同じで，異なるのはリード（$\overline{RD}$）信号が出ずに，ライト（$\overline{WR}$）信号が出る点と，CPU からデータバスへデータが乗せられる点です．

ライト（$\overline{WR}$）信号は，メモリリクエスト（$\overline{MREQ}$）信号より 1 クロック遅れて出されます．これは，データバスが安定するのを待つためで，ライト（$\overline{WR}$）信号でメモリの読み込みのゲートをコントロールすることができるようになっています．

メーカの立場とユーザの立場

マイクロプロセッサは，一つの電子部品との見方からすると，その使い方は，ユーザの責任において研究されなければなりません．汎用の IC やトランジスタでも，メーカの発表するのは一使用例にすぎず，回路構成や応用製品の信頼性については，すべてユーザの研究や雑誌などのレポートを参考にして使用されてきました．

コンピュータと呼ばれることから，大型コンピュータ並の指導やサービスをメーカに期待すると，あてがはずれます．LSI の単価には，これらの費用は含まれてはいないからです．マニュアルも有償である場合がほとんどです．マニュアルは，指導書ではありませんので，わかりにくいものです．基本的に「何をどうするとどうなります」という LSI の機能を述べ連らねてあるにすぎません．書物や雑誌に目を向け，いろいろな事例に接することが大切です．

* 命令により例外はある．

メモリライトサイクル

T_1　T_2　T_3

クロック

アドレスバス　メモリ アドレス

$\overline{MREQ}$

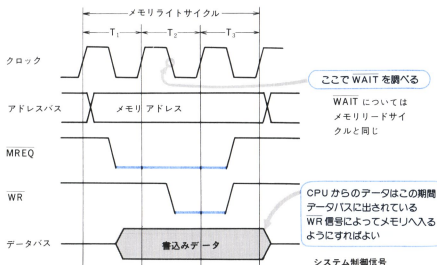
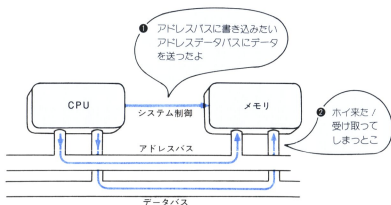
$\overline{WR}$

データバス　書込みデータ

ここで $\overline{WAIT}$ を調べる

$\overline{WAIT}$ については
メモリリードサイ
クルと同じ

CPU からのデータはこの期間
データバスに出されている
$\overline{WR}$ 信号によってメモリへ入る
ようにすればよい

システム制御信号

メモリライトサイクル
$\overline{MREQ}$ と
$\overline{WR}$ がアクティブ

30

IO リードサイクル，IO ライトサイクル

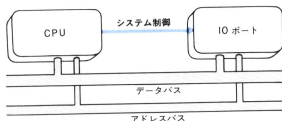
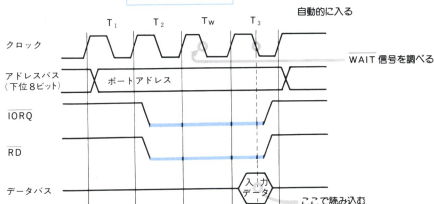
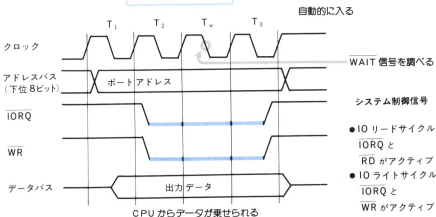
IO ポートに対するリード，ライトは，アドレスバスの下位8ビットが有効なアドレスとして使われます．上位8ビットへはなんらかの状態が出力されますので，無視しなければなりません．

イン〔IN〕命令，アウト〔OUT〕命令と，これらのくり返しの命令を実行したときに，このサイクルへ入ります．

IO サイクルでは，メモリの読み，書きより長く時間を必要とすることが想定されますので，ウエイトサイクル Tw が自動的に入ります．もっと長く時間待ちさせたいときは，この Tw の立ち下がりの前後にウエイト ($\overline{WAIT}$) 信号を入れることになります．またリード ($\overline{RD}$)，ライト ($\overline{WR}$) および IO リクエスト ($\overline{IORQ}$) の各信号は，立ち下がり，立ち上がり共に同じタイミングです．

イン〔IN〕命令，アウト〔OUT〕命令は，入出力するポートアドレスを指定しなければなりません．直接指定の場合は A レジスタとの間でのやりとりになり，ポートアドレスを n とすると〔IN　A，(n)〕，〔OUT　(n)，A〕と書きます．間接指定の場合は，ポートアドレスを C レジスタに入れておき，〔IN　r，(C)〕，〔OUT　(C)，r〕(ここで r は A，B，C，D，E，H，L のいずれかのレジスタ) と書きます．このほかにメモリの一連の複数バイトのブロックを入出力するブロック入出力命令があります．

システム制御信号

- IO リードサイクル
  IORQ と
  RD がアクティブ
- IO ライトサイクル
  IORQ と
  WR がアクティブ

31 リフレッシュサイクル

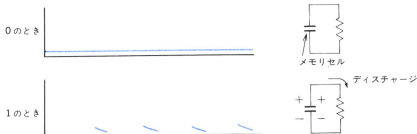
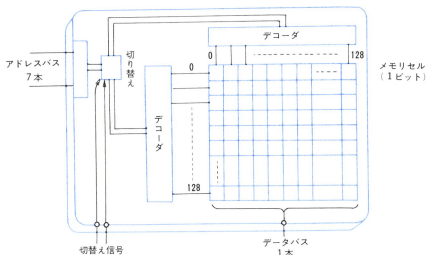
リフレッシュサイクルは，各フェッチサイクルの後半に行われます．アドレスバスの下位7ビットには，Rレジスタの内容が出力されます．Rレジスタの内容は1回ごとに1ずつ増やされ，DRAMの1列ごとに再書き込みを行ないます．

このときはリード($\overline{RD}$)，ライト($\overline{WR}$)の信号の代わりにリフレッシュ($\overline{RFSH}$)信号が出て，リフレッシュサイクルであることを知らせます．

リフレッシュは，長くても2ms以内に1回以上行なわなければならないので，Z-80を最高速で働かせたときでも，16KビットのDRAMが限界になります．16KビットのDRAMの代表的なものに，4116と呼ばれるものがありますが，16 K=2^{14}で，アドレスバスは14ビット必要です．ところが上位，下位7ビットずつを時分割して与えますので，アドレス入力は7端子しかありません．リフレッシュの場合は，7ビットのアドレスだけを順次与えればよいのです．したがって，Rレジスタは7ビットとなっています．

この機能はZ-80が開発されたころ，主流であった16KビットDRAMを対象に考えられたため，現在使われるメガビットクラスのDRAMでは，Z-80の機能とは別にリフレッシュ回路を作らなければなりません．

MOS LSIの取り扱い

Z-80ファミリに限らず，MOSプロセスのLSIは，端子のインピーダンスが高いため帯電しやすく，チップの絶縁が破壊することがあります．保存するときはモスパッドと呼ばれる導電性のスポンジにさしてピン間をショートしておけばまちがいがありません．また，ピンに直接手で触れたり，帯電していそうな所に接触しないように注意しなければなりません．はんだごても絶縁のよいものか，アースのできるものを使用します．実際には，そう簡単にこわれてしまうわけではありませんが，念のために心がけたほうがよいでしょう．

●アドレスバス7本へ　上位7ビット，下位7ビットを時分割で与える．

●いま，上位が与えられているか，下位が与えられているかを切替え信号で知らせてやる．

●リフレッシュの場合は，一方だけを与え，他方は全メモリセルをセレクトする．→ データ出力は無意味（デタラメ）→ データ出力なし．

32

CPUと周辺の接続（インターフェース）

各マシンサイクルごとにアドレスバスとデータバスの意味が変わりますので，どの動作をしているかによって，メモリ空間とIO空間のどれを選ぶか，また，書き込みか読み出しかの切り換えをしなくてはなりません．

まず，アドレスバスをデコードして，セレクト信号を作ります．次にメモリリクエスト（$\overline{\mathrm{MREQ}}$）とIOリクエスト（$\overline{\mathrm{IORQ}}$）信号によりメモリ空間とIO空間を切り替えます．ここでIOリクエスト（$\overline{\mathrm{IORQ}}$）信号はエムワン（$\overline{\mathrm{M1}}$）信号と同時にアクティブになったときは別の意味（割り込み応答）になりますので，このときはIO空間をセレクトしてはいけません．Z-80ファミリではセレクトしても問題ありません．

IOポートにZ-80ファミリのLSIを使うときは，同じ名称のピンどうしを接続すれば，システム制御信号に関してはなんら心配はいりません．2個以上のLSIを接続するならば，アドレスデコードしたチップセレクト信号をそれぞれのLSIに与えなければなりません．

メモリは種類によって異なります．ROMの場合は，チップセレクト信号は，リード（$\overline{\mathrm{RD}}$）信号が出たときだけ与えるようにしなければなりません．RAMの中にはアウトプットドライブ（$\overline{\mathrm{OD}}$）端子があるものがありますが，この端子へはリード信号を直接与えます．ライトイネーブル（$\overline{\mathrm{WE}}$）端子へはライト（$\overline{\mathrm{WR}}$）信号を与えればよいのです．アウトプットドライブ（$\overline{\mathrm{OD}}$）端子がないRAMの場合は，ライトイネーブル（$\overline{\mathrm{WE}}$）端子へ与える信号には工夫が必要です．メモリライトサイクルでは，メモリリクエスト信号が出て少したってからライト信号が出ますが，このライト（$\overline{\mathrm{WR}}$）信号が出るまでの時間，メモリは読み込み動作をしてしまい，CPUからのデータとメモリからのデータがデータバス上でぶつかってしまいます．実際問題としてはLSIをこわしてしまうほどのことはないようですが，さけるべきです．リード（$\overline{\mathrm{RD}}$）信号とリフレッシュ（$\overline{\mathrm{RFSH}}$）信号がなく，かつ，メモリリクエスト（$\overline{\mathrm{MREQ}}$）信号が出たときだけライトイネーブル（$\overline{\mathrm{WE}}$）端子へ与える信号を作れば解決できます．

システム制御信号

	$\overline{M1}$	$\overline{MREQ}$	$\overline{IORQ}$	$\overline{RD}$	$\overline{WR}$	$\overline{RFSH}$	
フェッチサイクル							メモリ
メモリ　リード							
メモリ　ライト							
IO　リード							IOポート
IO　ライト							
リフレッシュ							ダイナミックメモリ
割り込み応答							IOポート
		メモリ	IOポート	データの方向			
		アドレス空間					

●この信号の組合せでメモリやIOポートをイネーブル（活かす）したり，ディスエーブル（殺す）したりする（ように接続する）．
アドレスデコーダをコントロールして，セレクト信号が出るようにしたり，止めたりしてもよい．

●Z-80ファミリで構成するときは，同じ信号線どうしを接続すればよい．

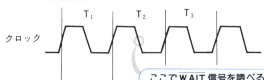
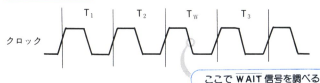
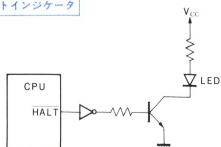
33

ウエイト信号とホルト信号

ウエイト（$\overline{\text{WAIT}}$）**信号**は，メモリやIOの応答速度がCPUのタイミングに間に合わないときに，ダミーのクロックサイクルを入れて，CPUを待たせるときに与えます．CPUは各マシンサイクルの2番目のクロックの立ち下がりの時点で，ウエイト信号端子の状態を検知して，このとき“L”になっていれば，次のクロックをウエイトサイクルとして何もしないで待ちます．このウエイトサイクルの立ち下がりのときも同じく検知していますので，“H”になるまで空転していることになります．ウエイト中はリフレッシュは行なわれません．

IOライトサイクルとIOリードサイクルでは一つ，割り込み応答のフェッチサイクルでは二つのウエイトサイクルが自動的に挿入されます．

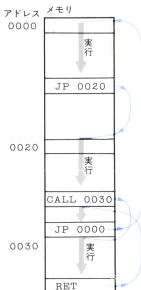
簡単なシステムではウエイトの端子を使用しないことのほうが多いと考えられますが，そのときはV_{CC}へ続いて“H”レベルにしておかないと，CPUは動作したり止まったり，不安定な状態になってしまいます．

ホルト（$\overline{\text{HALT}}$）**信号**は，CPUがホルト〔HALT〕命令を実行したときに出される信号です．〔HALT〕命令を実行すると，プログラムカウンタの進行を止めてしまいますので，CPUが停止したのと同じです．普通のプログラムでは，このようなことは起こり得ませんので，異常があったことを外部へ知らせる信号として使えます．ホルト状態からの解除は，リセット信号を与えるか，割り込み受け付け可になっているときは，割り込みをかければよいのです．ホルト状態では，ノンオペレーション〔NOP〕命令といって，フェッチサイクルだけで何もしない命令をくり返し実行しています．このようにしてある理由は，ホルト中もメモリリフレッシュを絶やさないためです．

- T_2の立ち下り，またはT_Wの立ち下りで$\overline{\text{WAIT}}$を調べ，“H”ならそのまま，“L”ならT_Wを挿入する．
- 挿入されたT_Wの立ち下りでも$\overline{\text{WAIT}}$を調べ，“H”なら次はT_3，“L”ならもうひとつT_Wを挿入する．
- $\overline{\text{WAIT}}$が“H”になるまで何回でもT_Wが入る．

34 割り込みの概念

CPUは，リセット(RESET)信号が入ると，ゼロ番地から順次命令を実行します．命令の中に，「ある番地へジャンプせよ」という命令，ジャンプ〔JP〕命令やコール〔CALL〕命令があると，その番地の命令を実行します．つまり，通常はプログラムカウンタを一つずつくりあげていき，特別の命令があると，プログラムカウンタの内容をその命令に従って変更します．

いま，このようにして処理を進めているときに，急に他の処理をしなければならなくなったことを考えてみます．停電になってとりあえずバッテリに切り換えるとか，端末装置から故障を知らせる信号が入ったときに，通常のプログラムの実行を止めて，対策処理をしなければならないことがあります．このようなときに，インタラプト（INT）信号かノンマスカブルインタラプト（NMI）信号を入れることによって，プログラム上の命令とは関係なく，特定の番地へジャンプさせることができます．これを**割り込み**と呼びます．

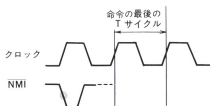
割り込み機能を積極的に使うことにより，処理効率を高めたり，プログラムを簡単にしたりすることができます．たとえば，CPUからプリンタにデータを打ち出すよう命令を与えてからプリンタが動作を終えるまでの時間は，CPUの処理速度に対して非常に長いのですが，この間ただ待つのではなく，CPUは別の仕事をしていて，プリンタからの動作終了割り込みで次のデータを打ち出すことができます．一度に二つの仕事をしているように見えるのでマルチタスクとかマルチジョブと呼んで，入出力の多い仕事ではCPUの処理時間は無視され，端末装置の動作時間だけで処理速度がほぼ決まってきます．

割り込みはCPUの動作に同期せずに，外部からのハード的な要求で呼び出されるサブルーチンコールと考えてよいでしょう．

通常の実行
アドレス
メモリ
0000
実行
JP 0020
0020
実行
CALL 0030
JP 0000
0030
実行
RET
●普通は順序よく.
●ジャンプ，コールなどの命令があれば指定の番地へ実行を移し，そこから実行する.
命令により実行アドレスを変える
割り込み
実行
ある番地
電気信号（NMI か INT 端子）割り込み信号
●ある番地の命令を実行中（命令は何でもよい）電気信号が入ると特別の番地へ実行を移す．RETI（リターン命令）があると元の命令の次（さっきのつづき）へ実行を戻す.
特別の番地
実行
RETI
割り込み処理ルーチン
電気信号により実行アドレスを変える

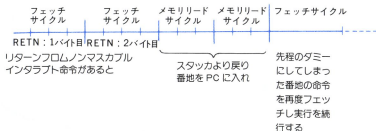
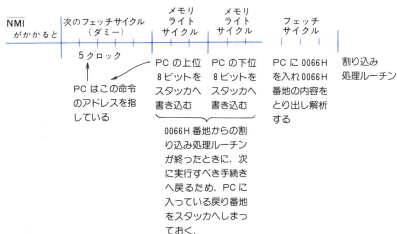
35

ノンマスカブルインタラプト（$\overline{NMI}$）

Z-80には，ノンマスカブルインタラプト（$\overline{NMI}$）と，ただのインタラプト（$\overline{INT}$）の2系統の割り込みがあります．ノンマスカブルインタラプトは，いかなるときでも受け付ける最優先の割り込みです（ただし，バスアクノリッジ中は受け付けません）．したがって，停電などシステムやオペレータにダメージを与えるような非常時の対策に使うことが多いようです．

ノンマスカブルインタラプトは名前のとおり，プログラムによってマスク，すなわち無効にすることができない割り込みです．CPUはノンマスカブルインタラプト（$\overline{NMI}$）信号が立ち下がったとき，実行中の命令が終わり次第，割り込み処理に入ります．割り込みの処理プログラムが終わったとき，また元の番地へ戻らなければならないために，現在のプログラムカウンタ（PC）の内容をスタッカへ格納します．次に，プログラムカウンタへ0066Hを書き込み，新しいフェッチサイクルに入ります．したがって，0066H番地をコールすることになります．割り込みがかかった時点で実行中の命令は終わりまで実行し，次の命令のフェッチサイクルもあたかも平常のとおりに実行されますが，データバスを無視して〔CALL 0066H〕をフェッチしたかのように動作を続けていると考えられます（ただし，タイミングは異なります）．0066H番地には割り込み処理プログラムを書いておかなければなりません．そして処理の最後にリターンフロムノンマスカブルインタラプト〔RETN〕命令があれば，スタッカへ格納しておいた戻り番地をとり出して，割り込みがかかったときに実行していた次の命令へ復帰します．ノンマスカブルインタラプト（$\overline{NMI}$）が入ってからリターンフロムノンマスカブルインタラプト〔RETN〕が実行されるまではインタラプト（$\overline{INT}$）は無視され，待たされることになります．

これより前に $\overline{NMI}$ の立ち下りがあると

次のフェッチサイクルは読み込んだ命令を無視してプログラムカウンタの内容をスタッカへ入れ0066H 番地へジャンプする

36

インタラプト（$\overline{INT}$）

リセット信号が入ってCPUが初期状態から実行し始めると，インタラプト信号に対するマスクは，ディスエーブルすなわち割り込みを無視するようになっています．また割り込みモードは0になっています．もし割り込みモード1か2で使用するのなら，〔IM1〕か〔IM2〕の命令を実行させておく必要があります．割り込みモード2の場合は，Iレジスタとペリフェラル内のレジスタに対し数値を設定しなければなりません．これらの準備が終わったところで，割り込みを許可する命令，イネーブルインタラプト〔EI〕命令を実行させますと，これ以降割り込みがかかるようになります．割り込みを禁止するときは，ディスエーブルインタラプト〔DI〕命令です．

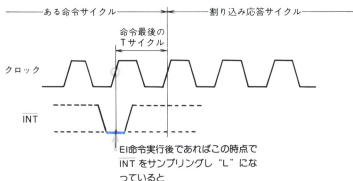
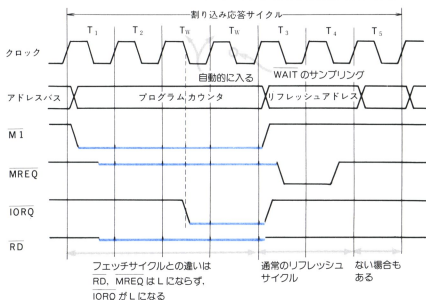
割り込みがかかると，次の割り込みがかからないようにディスエーブルインタラプト〔DI〕命令を自動的に実行し，おのおのの番地へジャンプします．割り込み処理プログラムの終わりには，リターン〔RET〕命令が書かれていれば，CPUは元の処理を続行するべく戻り番地へジャンプします．ただしこのとき，リターン〔RET〕命令の直前にイネーブルインタラプト〔EI〕命令を置かなければ，割り込みは禁止されたままになってしまいます．イネーブルインタラプト〔EI〕はただちに有効になるのではなく，次の1命令を実行し終わったときから有効になりますので，リターン〔RET〕命令が終了した後に割り込みを受け付けるようになります．

なお，ノンマスカブルインタラプトは信号の立ち下がりエッジでかかりますが，インタラプトのほうは各命令の最後のクロックサイクルの立ち上がり時点で信号の状態を検査しますので，このとき“L”になっていなければなりません．

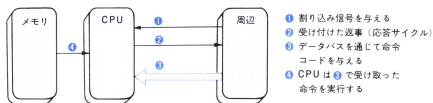
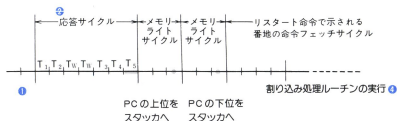
Z-80ファミリのペリフェラルは，優先順位を決定する機能を個々に持っており，外部に回路を必要としません．この機能を使うときは，割り込み処理が終わったことを通常のサブルーチンからのリターンとは区別してペリフェラルに与えなければならないため，リターン〔RET〕命令ではなくリターンフロムインタラプト〔RETI〕命令を使用します．

IM 0 IM 1 IM 2	割り込みモードを設定する命令．いずれかのモードを選びプログラムの最初の方で1回実行させる．
EI	割り込み信号を許可する命令
DI	CPU 側で割り込み信号を受け付けなくする命令

EIを実行後 CPU は割り込みを受け付け，DIを実行すると受け付けなくなる．

37

モード0のインタラプト

割り込みモード0は，命令の最後のクロックサイクルの立ち上がりのとき，インタラプト（$\overline{\mathrm{INT}}$）信号が“L”になっていると，次のフェッチサイクルは割り込み応答サイクルとなります．フェッチサイクルとの違いは，メモリリクエスト（$\overline{\mathrm{MREQ}}$）信号が出ずにIOリクエスト（$\overline{\mathrm{IORQ}}$）信号が出力されることと，3，4番目のクロックサイクルにウエイトサイクルが2個，自動的に挿入されていることです．したがって，ペリフェラル側では，エムワン（$\overline{\mathrm{M1}}$）信号とIOリクエスト（$\overline{\mathrm{IORQ}}$）信号が共にアクティブ，すなわち“L”になったことで，割り込みがかかったことを知ります．割り込み応答サイクルでは，アドレスバスには次の命令のアドレス，すなわちプログラムカウンタの値が出ますが，外部からはこれとは関係なく，なんらかの命令をデータバスに乗せてやります．この命令は主にリスタート〔RST〕命令か，コール〔CALL〕命令を使います．リスタート〔RST〕命令は1バイト構成ですからこれでよいのですがコール〔CALL〕命令のときは3バイト構成です．そのため，あと2回メモリリードサイクルに合わせて，データバスへ割り込み原因に応じたプログラムルーチンを実行させるための命令の続き（コール命令のジャンプ先番地を意味するオペランド）を与えなければなりません．

CPUの読み込みタイミングは，フェッチサイクルと同じくT_3の立ち上がりです．またリスタート〔RST〕命令を与えたときは，スタックポインタの変更をするためのクロックサイクルT_5が付きますが，コール〔CALL〕命令ではこれがなく，マシンサイクルの3番目（M3サイクル）が，メモリリードサイクルの3クロックの後にT_4として続き，4クロックサイクルとなります．これは通常の場合にフェッチサイクルでコール〔CALL〕命令やリスタート〔RST〕命令をフェッチしたのと同じです．すなわち応答サイクルで，アドレスバスを無視して強制的にデータバスへ命令を与えれば，それ以後は通常その命令を実行するのと同じ動作をしています．

メモリ
CPU
周辺
❶ 割り込み信号を与える
❷ 受け付けた返事（応答サイクル）
❸ データバスを通じて命令コードを与える
❹ CPU は❸で受け取った命令を実行する
リスタート〔RST〕命令を与えた場合
応答サイクル
メモリライトサイクル
メモリライトサイクル
リスタート命令で示される番地の命令フェッチサイクル
T1 T2 Tw Tw T3 T4 T5
割り込み処理ルーチンの実行❹
PCの上位をスタッカへ
PCの下位をスタッカへ
この時点でCPUはデータバス上の命令を読み込む（RST 命令）❸
コール〔CALL〕命令を与えた場合
応答サイクル
メモリリードサイクル
メモリリードサイクル
メモリライトサイクル
メモリライトサイクル
コール命令で示される番地の命令フェッチサイクル
T1 T2 Tw Tw T3 T4 T1 T2 T3 T1 T2 T3 T4
割り込み処理ルーチンの実行❹
PCの上位をスタッカへ
PCの下位をスタッカへ
コール命令の3バイト目（ジャンプ先番地の上位）を読み込む
コール命令の2バイト目（ジャンプ先番地の下位）を読み込む
この時点でCPUはデータバス上の命令を読み込む（CALL 命令）❸
このタイミングに合わせて周辺からデータバスに乗せる

38

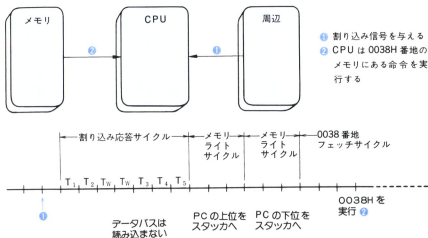
モード 1 のインタラプト

割り込みモード 1 は最も簡単な割り込みで，ほとんどノンマスカブルインタラプトと同じ動作をします．

インタラプト（$\overline{\mathrm{INT}}$）信号の検知と，応答サイクルは割り込みモード 0 と同じです．応答サイクルでは，外部からは何も与えず，CPU も何も読み込まないので，ダミーサイクルとなります．次に現在のプログラムカウンタの値，すなわち戻り番地をスタックへ格納するためにメモリライトサイクルが 2 回続き，そのあとプログラムカウンタを 0038H にします．次はこの番地の命令の実行サイクルへ入ります．見かけ上は，〔CALL　0038H〕を実行したことになります（ただし，タイミングは異なります）．

このモードでは，割り込み要因によって，ジャンプ先の処理ルーチンを分けることはできません．もしこれが必要なときは，0038H 番地からのルーチンの中で，外部の状況を読み込み，解析しなければなりません．

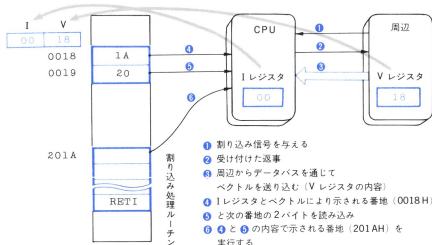
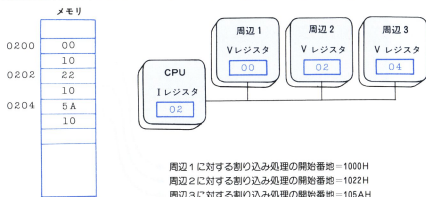
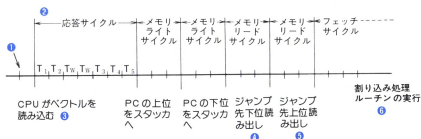
メモリ
CPU
周辺
❶ 割り込み信号を与える
❷ CPUは0038H番地のメモリにある命令を実行する
割り込み応答サイクル
メモリライトサイクル
メモリライトサイクル
0038番地フェッチサイクル
T1 T2 TW TW T3 T4 T5
データバスは読み込まない
PCの上位をスタックへ
PCの下位をスタックへ
0038Hを実行 ❷

39

モード2のインタラプト

割り込みがかけられた次のフェッチサイクルは，割り込み応答サイクルとなります．他のモードと違うのは，エムワン（$\overline{M1}$）信号とIOリクエスト（$\overline{IORQ}$）信号が共に“L”になったら，ペリフェラルからはベクトルと呼ばれる1バイトの値をデータバスに乗せ，CPUがこれを受け取ることです．次に戻り番地として現在のプログラムカウンタの値をスタックへ格納するためのメモリライトサイクルが2回続きます．その次のサイクルでは，CPUは2バイトの情報をメモリから読み込むためのメモリリードサイクルが2回続きます．そしていま読み込んだ値をプログラムカウンタへ入れて，そのアドレスからの実行を開始します．

ベクトルとは，各ペリフェラルにあらかじめ書き込んである1バイトのデータで，これがアドレスの下位8ビットとなります．上位8ビットは，CPU内のIレジスタへ，これもあらかじめ書き込んでおきます．割り込みがかかると，Iレジスタの内容とベクトルで構成される番地と，次の番地のメモリの内容を2バイト読み込んで，この値の番地へジャンプすることになります．

ベクトルの値は，各ペリフェラル個々に異なった値を書いておけば，割り込みをかけたペリフェラルによって，別々の処理ルーチンへジャンプしますし，DMAやSIOでは，割り込み発生の原因によってベクトルを変化させて送り出しますので，処理ルーチンの組み立てが簡単にできます．

このモード2の割り込み機能はZ-80の大きな特徴です．ファミリと合わせてシステムを構成すれば，外部回路なしで優先順位の決定を含めた，割り込みシステムが完成してしまいます．

周辺1に対する割り込み処理の開始番地＝1000H
周辺2に対する割り込み処理の開始番地＝1022H
周辺3に対する割り込み処理の開始番地＝105AH
このような表を割り込み処理ルーチンエントリーアドレステーブルと呼ぶ

40

デージーチェーン

割り込みの優先順位の決定にはデージーチェーン（ひな菊の輪）という手法がとられます．CPU に対して割り込み信号を出すペリフェラルを直列につないでおき，割り込みをかけて処理ルーチン実行中のペリフェラルが，自分より下位のペリフェラルに対して割り込み禁止を順次伝達していきます．自分より上位の割り込みがかかると，自分の処理が保留されますので，これも，デージーチェーンを通じて知ることができます．処理が終わってリターンフロムインタラプト〔RETI〕命令を CPU がフェッチすると，データバスを横から監視している処理中のペリフェラルは自分の処理が終わったことを知り，下位のペリフェラルに対して割り込みの禁止を解除します．

各ペリフェラルはインタラプトイネーブルイン（IEI）とインタラプトイネーブルアウト（IEO）信号の端子を持っており，IEO を自分より下位の IEI につなぎます．最も優先順位の高いペリフェラルの IEI は V_{CC}（+5V）へつなぎ，最も低いペリフェラルの IEO はオープンにしておきます．

多数のペリフェラルをつなぐときは，伝達に時間がかかりますので，工夫がいります．そのままでつなげるのは 4 個までです．

この方式は，割り込みだけでなく，マルチ CPU としたときや，**DMA** を複数使用するときのバスリクエストでも使われます．

デージーチェーン端子（入力＝IEI　出力＝IEO）の機能

① 通常IEIが“H”になっていればIEOも“H”が出ている．
② 割り込みを発生し（$\overline{INT}$）応答サイクルに入ると，割り込みを発生したペリフェラルはIEOを“L”にする．
③ IEIに“L”が入るとIEOも“L”にする．
④ 割り込みは，IEIが“H”になっているときだけ発生し，“L”のときは発生しない．またすでに発生した後にIEIが“L”になったときは保留されたものとする．

41

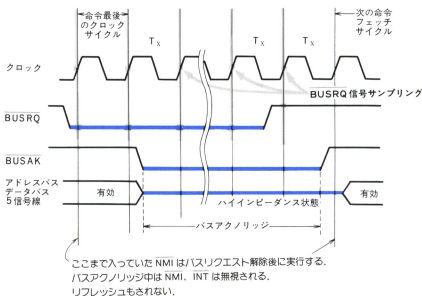
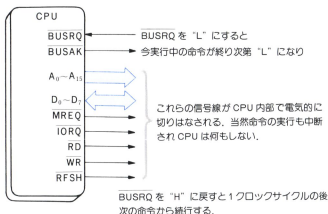
バス要求と応答

通常CPUはバスを占有して動作していますが，マルチCPUシステムで他のCPUからの要求や，その他いろいろな理由で一時動作を中止してバスを空けたいことがあります．このようなとき**バスリクエスト**（$\overline{\text{BUSRQ}}$）**信号**を与えると，トライステートの端子をすべて高インピーダンスにし，**バスアクノリッジ**（$\overline{\text{BUSAK}}$）**信号**をアクティブにして，外部に対してバスを空け渡したことを知らせてきます．この間メモリリフレッシュは行われなくなりますので，長時間この状態を続けるときは注意が必要です．

バスリクエスト（$\overline{\text{BUSRQ}}$）信号は，割り込み信号と同様に，命令の最後のクロックサイクルの立ち上がりの時点で検知されます．ここでバスリクエスト（$\overline{\text{BUSRQ}}$）が“L”であると，次の命令のフェッチサイクルはなくなり，バスアクノリッジ（$\overline{\text{BUSAK}}$）がアクティブになります．これ以後各クロックの立ち上がりでもバスリクエスト（$\overline{\text{BUSRQ}}$）が監視され“H”に戻っていると，次のクロックサイクルから元の動作に戻ります．バスアクノリッジ（$\overline{\text{BUSAK}}$）がアクティブな期間は，ノンマスカブルインタラプトとインタラプトは受け付けられなくなります．しかしバスリクエスト（$\overline{\text{BUSRQ}}$）が最初に検知される時点での両割り込みは受け付けられ，バスアクノリッジ（$\overline{\text{BUSAK}}$）の解除後に，それぞれの動作に入ります．

バスリクエスト（$\overline{\text{BUSRQ}}$）端子を使用しないときは，V_{CC}へつないで“H”レベルにしておきます．

命令最後のクロックサイクル
T_X
T_X
T_X
次の命令フェッチサイクル
クロック
BUSRQ 信号サンプリング
BUSRQ
BUSAK
アドレスバス
データバス
5信号線
有効
ハイインピーダンス状態
有効
バスアクノリッジ
ここまで入っていた NMI はバスリクエスト解除後に実行する．
バスアクノリッジ中は NMI，INT は無視される．
リフレッシュもされない．

42

内部レジスタの構成

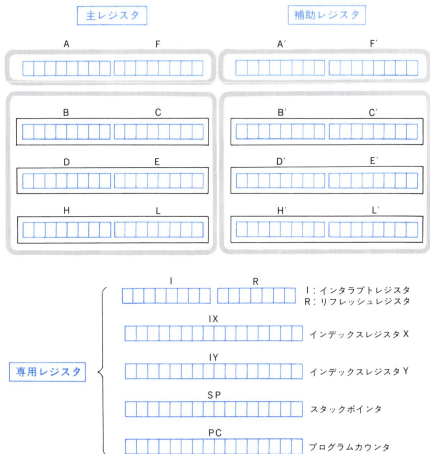
CPU 内部には，種々の作業に使うための一時的なメモリがたくさんあります．1ビット単独のものは，フリップフロップ（F/F）と呼ばれ，複数ビットが一連にして使われるものは，レジスタと呼ばれます．このうちでユーザに公開されるレジスタが22個あります．それぞれに個性を持っていて，うまく使い分けることが，よいプログラムを作るうえで重要です．どのレジスタをどう使うかは一概にはいえず，ケースバイケースで考えなければなりません．プログラムをいくつも作るうちに，だんだんとうまくなっていくでしょう．

8ビット単独で働くレジスタは，AレジスタとFレジスタです．Aレジスタはアキュムレータとも呼ばれ，最も重要な働きをします．Fレジスタはフラグレジスタとして各ビットに意味を持つレジスタです．8ビット単独でも，二つをつないで16ビットとしても働くレジスタは，B，C，D，E，H，Lレジスタです．ここまでの8個のレジスタは，主と補助の2組あり，内容を交換することができます．

専用レジスタとして用途が決められているものは6個あります．割り込み処理で使われるIレジスタは8ビット，メモリリフレッシュアドレスをカウントしているRレジスタは7ビットです．作表に便利なインデックスレジスタはIX，IYの2本あり，各16ビットです．スタッカのアドレスを入れておくスタックポインタとプログラム実行アドレスをカウントするプログラムカウンタも16ビットです．

16ビットレジスタには2桁，8ビットレジスタには1桁の名称が付けられており，補助レジスタには′（ダッシュ）を付けて，たとえばA′と呼ぶようになっています．

主レジスタ
補助レジスタ
A
F
A′
F′
B
C
B′
C′
D
E
D′
E′
H
L
H′
L′
専用レジスタ
I
R
I：インタラプトレジスタ
R：リフレッシュレジスタ
IX
インデックスレジスタX
IY
インデックスレジスタY
SP
スタックポインタ
PC
プログラムカウンタ

43

A, I, R, Fレジスタ

レジスタ群の中で最も多用されるAレジスタは，人が手作業で計算するときのソロバンにたとえられます．加算を行なう場合を例にとると，まず足される数をAレジスタへ入れます．次にAレジスタに対して，別の足す数を加えると，Aレジスタの内容が答になるわけです．このような用途のレジスタを**アキュムレータ**と呼びます．

8ビットの加算，演算，論理和，論理積，排他的論理和，比較は，すべてAレジスタの内容に対して行なわれます．

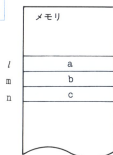
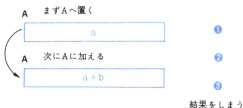
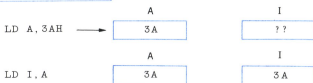
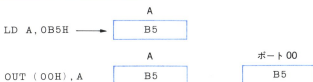
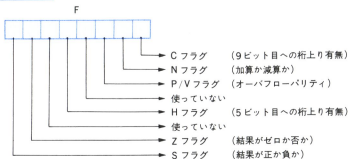
IレジスタとRレジスタに読み書きするときは，一度Aレジスタを経由しないとできません．入力，出力も，ポートアドレスに絶対番地を指定するときは，Aレジスタを経由します．

Iレジスタは，割り込みモードを2に指定したときに，割り込み処理ルーチンへジャンプするためのジャンプ先のアドレス指定のあるアドレスの上位8ビットを表わします．初期値設定のために1回書き込むだけで，ほとんどいじる必要はないレジスタです．

Rレジスタは，リセット信号でゼロになり，フェッチサイクルごとに1ずつ増加して，リフレッシュ用のアドレスを指定しています．プログラムで内容を変えることはほとんどありません．むしろ正確にリフレッシュさせるためには，さわらないほうがよいのです．

Fレジスタは，プログラムで内容を変えることはできません．1命令実行ごとに，その命令により出てきた結果に特別の意味があれば，それ以後の命令実行に必要な情報を記憶します．たとえば，減算命令を実行して結果がゼロであれば，Fレジスタ中の7番目のビット（これをゼロフラグといいます）を“1”にしてこのことを覚えておきます．次の命令が，「結果がゼロならばジャンプせよ」といった命令である場合，これをチェックして判断するようになっています．このように種々のフラグ（旗）が並んでいるレジスタがFレジスタです．

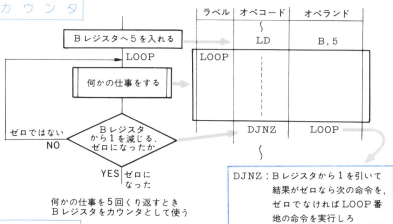
l番地の内容とm番地の内容を加えn番地へ格納する
A まずAへ置く
a
A 次にAに加える
a+b
❶
❷
❸
結果をしまう
メモリ
l
m
n
a
b
c
Iレジスタへ3AHを入れる
LD A, 3AH
A
3A
I
??
LD I, A
A
3A
I
3A
ポート00HへB5Hを出力する
LD A, 0B5H
A
B5
OUT (00H), A
A
B5
ポート00
B5
Fレジスタ
F
Cフラグ (9ビット目への桁上り有無)
Nフラグ (加算か減算か)
P/Vフラグ (オーバフローパリティ)
使っていない
Hフラグ (5ビット目への桁上り有無)
使っていない
Zフラグ (結果がゼロか否か)
Sフラグ (結果が正か負か)

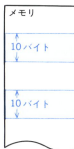
44

汎用レジスタ

汎用レジスタには，H，L，B，C，D，Eレジスタの6個と，同数の補助レジスタがあります．2個のレジスタをペアにして，HL，BC，DEと表現し，16ビットレジスタとして使うこともできます．

8ビットレジスタとしての働きは，主に演算途中の数値などを一時記憶することですが，Bレジスタはループ回数を入れておき，1回に1ずつ減算し，ゼロになったら，ループから出るというカウンタの役目をさせることがあります．このためにデクリメントジャンプノンゼロ〔DJNZ〕命令がとても便利です．Cレジスタは入力〔IN〕，出力〔OUT〕命令で，ポートアドレスを指定するポインタとして使えます．

16ビットレジスタとしてペアにして使うと，HLは加算，減算でアキュムレータとして働きます．また，アドレスを入れ，間接的にアドレス指定をすることができます．

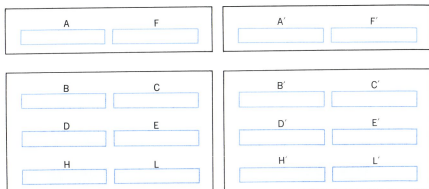
くり返し処理でアドレスを入れておくレジスタを**ポインタ**，回数を入れておくレジスタを**カウンタ**と呼びます．ブロック転送〔LDIR〕命令は，元のアドレスをHL，先のアドレスをDE，ブロックの長さ（バイト数）をBCに入れてから実行すると，HLの内容番地のメモリからDEの内容番地のメモリへ1バイト転送し，HLとDEに1を加え，BCから1を引きます．これをくり返しBCがゼロになると実行を終える，という命令です．HLとDEをポインタとして，BCをカウンタとして使用しています．

カウンタ
Bレジスタへ5を入れる
LOOP
何かの仕事をする
Bレジスタから1を減じる．ゼロになったか
ゼロではない
NO
YES
ゼロになった
何かの仕事を5回くり返すとき
Bレジスタをカウンタとして使う
ラベル
オペコード
オペランド
LD
B，5
LOOP
DJNZ
LOOP
DJNZ：Bレジスタから1を引いて
結果がゼロなら次の命令を，
ゼロでなければLOOP番
地の命令を実行しろ
ポインタ
Cレジスタの内容が00H
Aレジスタの内容が5AHとすると
OUT (C)，A
A
5A
C
00
ポート00
5A
Cの内容を番地とするポートへAの内容を出力せよ
ポインタとカウンタ
BCレジスタへ10を入れる
LD BC，10
HLレジスタへ0100Hを入れる
LD HL，0100H
DEレジスタへ0200Hを入れる
LD DE，0200H
HLの内容を番地とするメモリから
DEの内容を番地とするメモリへ
1バイト転送し
HLとDEにおのおの1を加え
BCから1を引け
BCがゼロになるまでくり返せ
LDIR
次の番地を示す
カウンタを減らす
メモリ
0100
10バイト
0200
10バイト
ブロック転送
そっくり転送し内容が同じになる

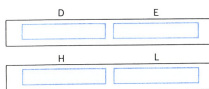
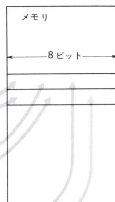
45

補助レジスタと交換命令

A, F, H, L, B, C, D, Eの各レジスタは，同じものがもう1組あります．**補助レジスタ**と呼び，主レジスタとは区別して，“′”ダッシュを付けて表現します．補助レジスタの内容は，交換命令と呼ばれる一連の命令で，主レジスタの内容と入れ換える以外に操作することはできません．したがって，主レジスタの裏側にあって，交換命令でひっくり返すと，表（主）へ出てくると考えられます．補助レジスタとの交換命令には，AFとAF′のペアを交換する〔EX AF, AF′〕と，上記以外の汎用レジスタを交換する〔EXX〕の二つがあります．

サブルーチンや割り込み処理ルーチンの中でメインルーチンで使用中のレジスタの内容を変えたくないことがあります．このようなときは，サブルーチンに入ったところで，レジスタの内容をメモリへ待避させてから，必要なレジスタを使い，サブルーチンから出る前に待避した内容をレジスタへ戻せばよいのですが，交換命令を使えば，1命令で全レジスタを待避させることができます．特に，スピードが問題になる割り込み処理の場合は都合のよい命令です．ただし，メインルーチンで主レジスタ，サブルーチンで補助レジスタを使うことをきちんと取り決めておかないと，どちらに何が入っているのかわからなくなります．

交換命令には他にDEとHLを交換する〔EX DE, HL〕と，HL, IX, IYの各内容とスタッカの内容を交換する〔EX (SP), HL〕，〔EX (SP), IX〕，〔EX (SP), IY〕があります．

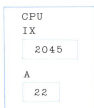
なお，通常の転送命令の場合は，たとえば〔LD A, B〕であれば，Bレジスタの内容がAレジスタへ転送されるだけで，元のAレジスタの内容はなくなってしまいます．Bレジスタは元のまま変わりません．交換命令では双方ともこわれることなく入れ換わります．

A
F
A′
F′
B
C
D
E
H
L
B′
C′
D′
E′
H′
L′
EXX
EX, EXX 以外の命令では
主レジスタしか扱えない
D
E
H
L
EX DE,HL
SP
とすると
メモリ
8ビット
H
L
n
n+1
EX (SP),HL
I X
EX (SP),IX
I Y
EX (SP),IY

46

IX，IY レジスタ

IX，IY レジスタは，**インデックスレジスタ**と呼ばれ，同一の働きをする 16 ビットレジスタが 2 本あります．通常はここへメモリのアドレスを入れて，間接的にアドレス指定をするときに使います．表を操作するときなどに表の先頭アドレスを入れておき，1 ずつ増加させながら次々に読み出すことができます．また，先頭から何番目のデータが欲しいというようなときは，先頭からの距離をプログラムで指定することもできるレジスタです．たとえば，〔LD A，(IX＋5)〕と書けば，IX レジスタに入っている数プラス 5 番地のメモリ内容を A レジスタへ入れることができます．距離は，－128～＋127 バイトまでの範囲に限られます．距離のことを**ディスプレイスメント**と呼びます．

このレジスタの内容を間接アドレスとしてソースまたはディスティネーションに指定した命令のマシン語は，オペコードが 2 バイト構成になります．3 バイト目には，ディスプレイスメントの値が入ります．オペランドがある場合は 4 バイト目に続きます．

実行のサイクルは，ディスプレイスメントの演算のためのクロックサイクルが入るため，例外的に長くなります．〔LD A，(IX＋5)〕では，19 クロックサイクルです．

たとえばIXレジスタに2045Hが入っていると

メモリ	
2045	0 0
6	1 1
7	2 2
	3 3

LD A,(IX+2)
により
Aレジスタには IX+2
すなわち2047H番地
の内容が転送される

LD A,(IX+2)のマシン語
DD オペコード
7E オペコード
02 ディスプレイスメント

メモリ	
2040	1
1	2
2	3
2043	1
4	2
5	3
2046	

メモリに3バイトを一組とするデータ
があるとする

```
          LD    IX,2040H      最初の組の先頭番地をIXへ
          LD    DE,3          DEへ3を入れておく
LOOP      LD    A,(IX)        1バイト目をAレジスタへ
          LD    B,(IX+1)      2バイト目をBレジスタへ
          LD    C,(IX+2)      3バイト目をCレジスタへ
                A,B,Cレジスタにある
                1組を処理する
          ADD   IX,DE         次の組の先頭番地をIXへ
          JP    LOOP
```

47 スタッカとスタックポインタ（SPレジスタ）

サブルーチンコール〔CALL〕命令を実行したり，割り込みルーチンへジャンプしたりするときに，戻り番地を自動的に記憶する機能があることは前述しました．これ以外にもプログラム中で一時的にレジスタの内容をメモリに退避したいことがよくあります．このようなときに大変便利なのがスタッカです．

スタッカはメモリ（RAM）の一部を割り当てた記憶領域のことですが，このアドレスは，**スタックポインタ**（SP）と呼ばれる16ビットレジスタに入れておきます．

スタックポインタに8025Hという値がプログラムによって入れられているとしましょう．このとき〔PUSH　HL〕を実行すると，8025H-1の8024HへHレジスタの内容が，8025H-2の8023HへLレジスタの内容が書き込まれ，スタックポインタは8023Hになっています．次に読み出すときは〔POP　HL〕を実行すると，8023Hの内容をLレジスタに，8023H+1の内容をHレジスタに転送し，スタックポインタは8025Hになります．

スタッカに次々におさめられた値は，後から入れたほうが先に読み出されますので，First In Last Out（FILO）タイプのスタッカと呼ばれます．サブルーチンや割り込みルーチンに入るときは自動的に〔PUSH PC〕* に相当する命令が実行され，戻るときは〔POP PC〕* に相当する命令が実行されていることになります．したがって，サブルーチンの中から次のサブルーチンをコールするときも，（これをサブルーチンの**ネスティング**といいますが）うまく働いてくれます．ただし，スタッカとして割り当てた領域を超えるネスティングはできません．未完成のプログラムで誤って次々にサブルーチンコールをすると，このようなことが起こることがあります．プログラム設計の際に，スタッカの大きさを決定するときは，慎重な判断を要します．またプログラムの始まりでは必ずスタックポインタのイニシャライズ（初期値設定）を行なってください．〔PUSH〕，〔POP〕できるペアレジスタはAF，BC，DE，HL，IX，IYです．

＊　この命令は，プログラム上では使えない．

CPU
SP
8025

メモリ（RAM）

アドレス	内容
801F	
8020	
8021	C
8022	B
8023	L
8024	H
8025	

スタッカ

PUSH　HL
により
8025H-1 へ H
8025H-2 へ L の内容が入る．
SP は 8023H になる

SP
8023

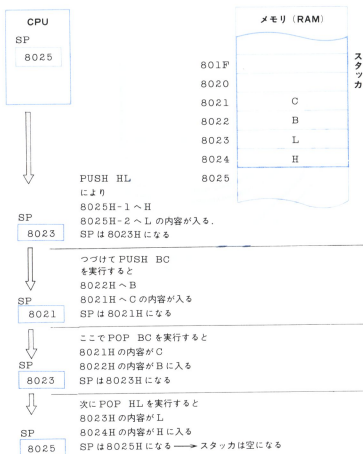
つづけて PUSH　BC
を実行すると
8022H へ B
8021H へ C の内容が入る
SP は 8021H になる

SP
8021

ここで POP　BC を実行すると
8021H の内容が C
8022H の内容が B に入る
SP は 8023H になる

SP
8023

次に POP　HL を実行すると
8023H の内容が L
8024H の内容が H に入る
SP は 8025H になる ──→ スタッカは空になる

SP
8025

プログラム中ではいちいち SP の値を気にしなくてもよい．

AA
PUSH　HL
PUSH　BC

ただし，PUSH と POP は対にして使わなければならない
（意図した変則的な場合もある）

この間で H，L，B，C の内容は変っても
AA での内容と BB での内容は同じになっている．

POP　BC
POP　HL
BB

CALL　では PUSH　PC
RET　では POP　PC　が実行されているのと同じなので，
かならず対にして使う

48

転送命令

最も基本的な転送命令は，ロード命令〔LD　a，b〕です．オペランドにはソースとディスティネーションの指定をします．ニモニックはZ-80ではすべて〔LD〕で統一されています．8ビットの転送では，ソースにはレジスタ名，ペアレジスタによる間接アドレス，インデックスレジスタによるディスプレイスメント，直接アドレス値および定数が指定できます．ディスティネーションには，ソースで使用できるうちの定数以外を指定できます．ただし，I，RレジスタはAレジスタとのやりとりしかできないなど，ソースとディスティネーションの組み合わせには限定があります．

16ビット転送では，ペアレジスタとやりとりのできる組み合わせは，16ビット定数，直接アドレス値だけで，ペアレジスタ間のやりとりは，HL，IX，IYレジスタからスタックポインタ（SP）への転送だけになります．

アセンブリ語で記述する場合，アドレス値や定数は，10進数，16進数で書いてもよいのですが，ラベル名で記述することもできます．マシン語に直したときは，アセンブルリスト上は，16進数に統一されます．16ビット定数やアドレス値のときは，オペコードの次に下位8ビット，その次に上位8ビットがくるように並びます．

レジスタや定数に（　）かっこを付けたオペランドは，メモリへまたはメモリからの転送命令を意味します．かっこ内の数値やラベルは，メモリのアドレスを指しています．16ビット転送の場合は，このアドレスが下位8ビット，次が上位8ビットの転送アドレスになり，ペアレジスタとの転送では，たとえばHLならば，Hレジスタが上位，Lレジスタが下位に対応します．

〔PUSH〕，〔POP〕も16ビット転送命令ですが，スタックポインタの項で述べました．

8ビットロード

ディスティネーション(受け側)		ソース(送り側) レジスタ	ソース(送り側) メモリ	定数
レジスタ	A	A, B, C, D, E, H, L, I, R	(HL), (BC), (DE), (IX+d), (IY+d), (定数)	定数
	B C D E H L	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
	I R	A	—	—
メモリ	(HL)	A, B, C, D, E, H, L	—	定数
	(BC) (DE)	A	—	—
	(IX+d) (IY+d)	A, B, C, D, E, H, L	—	定数
	(定数)	A	—	—

アドレス　　Aレジスタはすべての転送ができる　　メモリ→メモリの転送はできない　　8ビット定数

16ビットロード

ディスティネーション(受け側)		ソース(送り側) レジスタ	ソース(送り側) メモリ	定数
レジスタ	BC DE HL	—	(定数)	定数
	SP	HL, IX, IY	(定数)	定数
	IX IY	—	(定数)	定数
メモリ	(定数)	BC, DE, HL, SP, IX, IY	—	—

PUSH, POPのできるレジスタ
AF, BC, DE, HL, IX, IY

LD (nn), HL や
LD HL, (nn) の
場合
nn　　へL(下位)
nn+1へH(上位)
が対応する

	メモリ
nn	L
nn+1	H

49

算術演算命令

レジスタやメモリ内の8ビットないし16ビットのビットパターンを2進数とみなして，数学的な演算をする命令があります．8ビットの演算は，アド〔ADD〕，サブトラクト〔SUB〕，アドウィズキャリ〔ADC〕，サブトラクトウィズキャリ〔SBC〕の4命令があります．Aレジスタに対して，他のレジスタかメモリの内容を加減算します．この演算のとき桁あふれ，または桁借りをする命令と無視する命令に分かれます．あふれた桁はFレジスタの1ビット目のキャリフラグ（Cフラグ）があてられます．

コンペア〔CP〕命令は，Aレジスタからオペランドの内容を減算しますが，Aレジスタの内容は変わらず，Fレジスタが減算のときと同じに変化し，あとの命令で，結果がゼロであったかチェックできます．ゼロであったときは，Aレジスタの内容とオペランドの内容が同じであると判断できます．

インクリメント〔INC〕命令は，オペランドに1を加える命令です．デクリメント〔DEC〕は，オペランドから1を引く命令です．メモリ内のデータ列を1バイトずつ順次操作する場合やくり返し行なう仕事の回数をカウントするときに使います．

ニゲイト〔NEG〕は，Aレジスタの内容をゼロから引いてAレジスタに入れる命令です．すなわち，Aレジスタの内容の符号（プラス，マイナス）を反転させるとき主に使います．

16ビットの演算は，HL，IX，IYレジスタの内容に対して行なう加減算で，桁あふれを考えに入れない演算は加算だけです．インクリメント〔INC〕とデクリメント〔DEC〕は，8ビットと同様です．メモリアドレスの操作に使います．

算術演算命令でのFレジスタの働きは重要で，多桁の演算では，キャリフラグの働きに注意する必要があります．また10進補正〔DAA〕命令は，Fレジスタの結果を参照して補正のし方を決定します．

	命令	ディスティネーション 被演算数→答	ソース レジスタ	ー メモリ	ス 定数	操作 s：ソース Cy：キャリ
8ビット	ADD	A	A，B，C，D E，H，L	(HL) (IX+d) (IY+d)	定数	A+s→A
	ADC	A				A+s+Cy→A
	SUB	A*				A−s→A
	SBC	A				A−s−Cy→A
	CP	A*				A−s　A は不変
	INC	ソースと同じ*	A，B，C，D E，H，L	(HL) (IX+d) (IY+d)	—	s+1→s
	DEC	ソースと同じ*				s−1→s
	DAA	A*	A*	—	—	A を補正
	NEG	A*				0−A→A
16ビット	ADD	HL	BC，DE HL，SP	—	—	HL+s→HL
	ADD	IX	BC，DE SP，IX			IX+s→IX
	ADD	IY	BC，DE SP，IY			IY+s→IY
	ADC	HL	BC，DE HL，SP			HL+s+Cy→HL
	SBC	HL				HL−s−Cy→HL
	INC	ソースと同じ*	BC，DE，HL SP，IX，IY			s+1→s
	DEC	ソースと同じ*				s−1→s

＊印はアセンブリ語ではディスティネーションはオペランドに書かない．（例）SUB　C　　INC　A
DAA　NEG ではソースも書かない．

ADD　ADC

```
  10111001
+)01010101
 100001110
```

桁上り　C フラグ（キャリ）へ入る

ADC では C フラグ（キャリ）もここへ加える．すなわち，さらに下の桁の演算がこの前に行なわれているとすると，そのときの桁上りが含まれる

- SBC は，桁借りが C フラグに入っているのでこれを含めての演算をする．
- SUB は，前の演算で桁借りがなかったものとして C フラグは無視してしまう．

50

論理演算命令

論理演算命令は，すべて8ビット単位に，Aレジスタに対して行われます．各ビットごとの論理積，論理和，排他的論理和，否定をとります．

アンド〔AND〕命令は論理積です．Aレジスタとオペランドの内容の双方共に“1”のビットだけ“1”が残り，他は“0”になります．Aレジスタのある特定のビットだけ残したいときは，ここを“1”，他を“0”としたビットパターンをオペランドで指定して，アンドをとれば，不要なビットは必ず“0”になり，必要なビットだけが“1”であれば“1”，“0”であれば“0”として残ります．この方法を**マスク**といいますが，Z-80ではビット操作命令があるのであまり使いません．

オア〔OR〕命令は論理和です．Aレジスタとオペランドの内容のいずれかが“1”のビットを“1”として，双方共に“0”のビットを“0”にします．16ビットの演算のデクリメント〔DEC〕命令でHLレジスタの内容を1ずつ減らしてゼロになってもフラグは変化しません．そこで，このチェックを行なうときは，Hレジスタを A レジスタに移して，L レジスタとのオア〔OR〕をとりますと，H，L，共に全ビットが“0”，すなわちHLレジスタとしてゼロになっているときだけ結果がゼロになり，フラグレジスタのゼロフラグがこのことを示します．また，特定のビットを前の内容にかかわらず“1”にしたいときには，このビットが“1”，他が“0”のビットパターンとオア〔OR〕をとれば，“1”のところは“1”になり，“0”のところは前のまま残ります．

エクスクルーシブオア〔XOR〕命令は排他的論理和です．Aレジスタとオペランドの内容が，ビット単位に同じところを“0”，異なるところを“1”にします．AレジスタとAレジスタのエクスクルーシブオア〔XOR〕をとるとAレジスタは必ずゼロになります．なおキャリフラグだけを“0”にするときは，〔AND A〕か〔OR A〕を使います．

コンプリメント〔CPL〕命令はビットパターンをすべて反転させます．すなわち，“1”のビットは“0”に，“0”のビットは“1”にする論理否定です．

命令	ディスティネーション	ソース		
		レジスタ	メモリ	定数
AND OR XOR	A	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
CPL	A	A	—	—

アセンブリ言語ではディスティネーションはオペランドに書かない。
CPL ではソースも書かない。

AND

a	b	結果
0	0	0
0	1	0
1	0	0
1	1	1

（例）

Aレジスタ	0	1	0	0	1	0	1	1
ソース	1	1	0	1	0	0	0	1
Aレジスタ（結果）	0	1	0	0	0	0	0	1

OR

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	1

（例）

Aレジスタ	0	1	0	0	1	0	1	1
ソース	1	1	0	1	0	0	0	1
Aレジスタ（結果）	1	1	0	1	1	0	1	1

XOR

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	0

（例）

Aレジスタ	0	1	0	0	1	0	1	1
ソース	1	1	0	1	0	0	0	1
Aレジスタ（結果）	1	0	0	1	1	0	1	0

CPL

前	結果
0	1
1	0

（例）

Aレジスタ	0	1	0	0	1	0	1	1
Aレジスタ（結果）	1	0	1	1	0	1	0	0

51

ビット操作命令

レジスタ，メモリ内のどこの1ビットでも，ビット操作命令で“0”か“1”にすることができ，また“0”か“1”かの判定をすることができます．コントローラ的な応用にはよく使う命令で，Z-80の特徴の一つといえます．

セット〔SET〕命令は，ビットを“1”にする命令で，オペランドにはビットの位置とレジスタ名か，メモリのアドレスを示すレジスタ HL，IX，IY のいずれかを指定します．ビットの位置は，2の n 乗の n で表わしますので，右端が0，左端が7となります．

リセット〔RES〕命令は，ビットを“0”にする命令です．オペランドはセット命令と同じです．

ビット〔BIT〕命令は，指定のビットを調べ，“0”であればFレジスタのゼロフラグ（Zフラグ）を“1”にして，“0”であったことを記憶しておきます．後に書かれた条件付きジャンプ命令で，“0”のときと1”のときの飛び先番地を別々に指定してあれば，条件に応じた手順で仕事を進めます．オペランドはセット命令と同じです．

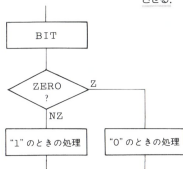
23A6H番地のメモリの右から3番目のビットを“1”にするときは，HLレジスタに23A6Hをロード命令で書き込んでから〔SET 2,(HL)〕を実行させ，“0”にするときは〔RES 2,(HL)〕，内容を判定するには〔BIT 2,(HL)〕となります．

命令	ディスティネーション	ソース			
		レジスタ	メモリ		ビット
SET RES BIT	ソースと同じ	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)		0～7

アセンブリ言語ではディスティネーションはオペランドに書かない．

SET　"1"にする

RES　"0"にする

BIT　指定のビットが"0"か"1"かによって
Fレジスタの Z フラグを
"0"なら Z（ゼロ）
"1"なら NZ（ノンゼロ）
を示すように設定する．
この場合，後にある判定命令（ジャンプ命令）でZフラグの状態を判定し以後の処理を分岐させる．

```
BIT  0,A      Aレジスタの0ビット
JP   Z,ZRUTIN
```

ゼロでなければここからの
プログラムを実行

52

ローテート，シフト命令

ローテート命令は，ビットパターンを右か左に一つだけずらす命令です．はみ出したビットは最後に回り込んでつながります．このときキャリフラグを含めて回転するか，含めずに回転し，回り込んだビットと同じものをキャリフラグへ入れるかによって大別されます．ローテートライト〔RR〕，ローテートレフト〔RL〕，ローテートライトサーキュラ〔RRC〕，ローテートレフトサーキュラ〔RLC〕があり，特にAレジスタに対する命令は，8080系のCPUと互換性を持たせるために，1バイト命令が別に用意されています．

ローテートライト（レフト）デジット〔RRD〕，〔RLD〕命令は，HLレジスタの内容で示されるメモリとAレジスタの下4ビットとの間で4ビット（1デジット）を一まとめにして回転させる命令で，2進化10進数の多桁演算によく使われるものです．

シフト命令は，はみ出したビットを切りすて，最後には“0”または左端ビットの値（符号ビット）が入ります．シフトライトアリスメチック（算術的右シフト）〔SRA〕，シフトレフトアリスメチック〔SLA〕，シフトライトロジカル〔SRL〕の三つです．算術的シフトは，ビットパターンを2進数とみなしたときシフトしても符号が変わらないよう配慮されているシフトです．左シフトは算術的（アリスメチック）も論理的（ロジカル）も同じですから一つしかありません．左に一つシフトすると，数値は2を掛けた値になり，右にシフトすると2で割った値になります．

ビットパターンを2進数扱いでかけ算，割り算をするときは，シフトとローテートを使って高速のルーチンを組むことができます．

	命令	ソース（ディスティネーション）レジスタ	メモリ	動作
ローテート	RLCA	A*	—	Cy ← 7 ← 0
	RLC	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	
	RLA	A*	—	Cy ← 7 ← 0
	RL	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	
	RRCA	A*	—	Cy ← 7 → 0
	RRC	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	
	RRA	A*	—	Cy → 7 → 0
	RR	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	
	RLD	A+(HL)*		A 7 4 3 0 (HL) 7 4 3 0
	RRD			A 7 4 3 0 (HL) 7 4 3 0
シフト	SLA	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy ← 7 ← 0 ← "0"
	SRA			Cy 7 → 0
	SRL			Cy "0" → 7 → 0

＊印はアセンブリ言語ではオペランドに記述しない　　Cy は F レジスタの C フラグ（キャリ）

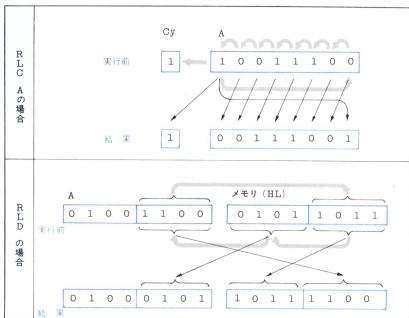
- SRA の7番ビットは変わらない．
- シフトではCy の元の値は失われる．
- RRD，RLD では移動する4ビット単位の内容は変わらない．

命　令	Cy	A
RLC　A	1	00111001
RL　A	1	00111001
RRC　A	0	01001110
RR　A	0	11001110
SLA　A	1	00111000
SRA　A	0	11001110
SRL　A	0	01001110

RLC Aの場合

Cy A
実行前 1 1 0 0 1 1 1 0 0
結　果 1 0 0 1 1 1 0 0 1

RLD の場合

A メモリ（HL）
実行前 0 1 0 0 1 1 0 0 0 1 0 1 1 0 1 1
結　果 0 1 0 0 0 1 0 1 1 0 1 1 1 1 0 0

シフト，ローテートを使ったプログラム例

(16ビットの掛け算サブルーチン)

メモリに図のようにaとbの値が2進数で入っている．

a×b=c

の演算をし，答cを32ビットとして，メモリに格納する．

メモリのアドレスnはIXレジスタに入れられているとする．

```
MPX   LD    C, (IX+0)
      LD    A, (IX+1)
      LD    E, (IX+2)
      LD    D, (IX+3)
      LD    B, 16
      LD    HL, 0
      EXX
      LD    HL, 0
      LD    DE, 0
      EXX
LOOP  SRL   A
      RR    C
      JR    NC, JNDT
      ADD   HL, DE
      EXX
      ADC   HL, DE
      EXX
JNDT  SLA   E
      RL    D
      EXX
      RL    E
      RL    D
      EXX
      DJNZ  LOOP
      LD    (IX+4), L
      LD    (IX+5), H
      EXX
      LD    (IX+6), L
      LD    (IX+7), H
      RET
```

	メモリ
n	a (下　位)
	a (上　位)
	b (下　位)
	b (上　位)
	c (最下位)
	c (下　位)
	c (上　位)
	c (最上位)

53

ブロック転送，ブロックサーチ，ブロック入出力命令

メモリ内の複数バイトのブロックを別の番地へ移し変える場合，1バイトずつCPU内のレジスタへ読み込み，別の番地へ書き込む操作を必要バイト数になるまでくり返します．また，メモリ内のブロックの中から指定のビットパターンと同じパターンを持つ1バイトを探し出すときも，1バイトずつAレジスタへ読み込んではコンペア〔CP〕命令を実行し，一致するまでくり返します．入出力のときもメモリブロックの内容をポートへ次々に出力したり，次々にポートから読んだデータをメモリへ並べたりするときはくり返しのプログラムを組まなければなりません．これを一つの命令で置き換えられるのが，この命令群です．

ブロック転送命令は，HLレジスタの元のアドレス，DEレジスタに転送先のアドレス，BCレジスタに転送バイト数をあらかじめ書いておき，次に転送命令を実行させますと，HLとDEレジスタの内容を1ずつ進め，BCレジスタの内容を1ずつ減らしていきます．

ブロックサーチ命令は，HLレジスタにアドレス，BCレジスタにブロックのバイト数を，比較すべきビットパターンをAレジスタに書いておきます．一致するとFレジスタのゼロフラグが，BCレジスタがゼロになる（一致するものがない）とオーバフローフラグ（Vフラグ）がリセットされますので，次の命令でフラグ判定をしなければなりません．

ブロック入出力命令は，HLレジスタにデータ格納のメモリアドレス，CレジスタにIOポートアドレス，Bレジスタにバイト数を書いてから実行させます．

これらの命令には，メモリブロックを先頭から扱うインクリメントグループと後から扱うディクリメントグループがあります．また，全データの操作が終わる（サーチでは一致した場合も含む）までプログラムカウンタが変わらないで，自動的にループするリピートの機能を持つ命令もあります．転送，サーチ，入出力しながら，何かのデータ操作がない場合は，リピートは便利です．1バイト扱うごとにフェッチサイクルから始まります．

ブロック転送

（HL←HL+1の意味は，HLの内容に1を加えるということ）

命　令	動		作
LDI	HLの内容を番地とするメモリから DEの内容を番地とするメモリへ転送する	HL←HL+1　BC←BC−1	1バイトだけ転送して終る
LDIR		DE←DE+1	BC=0までくり返す
LDD		HL←HL−1　BC←BC−1	1バイトだけ転送して終る
LDDR		DE←DE−1	BC=0までくり返す

BC≠0のときはVフラグがセットされる

ブロックサーチ

命　令	動		作	
CPI	HLの内容を番地とするメモリの内容とAレジスタの内容を比較する	A−(HL)=0…等しい A−(HL)≠0…等しくない 等しいときZフラグをセット（ゼロの状態）	HL←HL+1	1バイトだけ調べて終る
CPIR			BC←BC−1	BC=0か A−(HL)=0まで
CPD			HL←HL−1	1バイトだけ調べて終る
CPDR			BC←BC−1	BC=0か A−(HL)=0まで

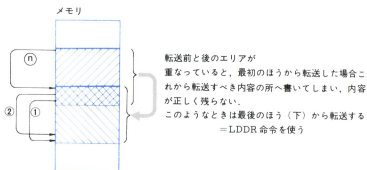
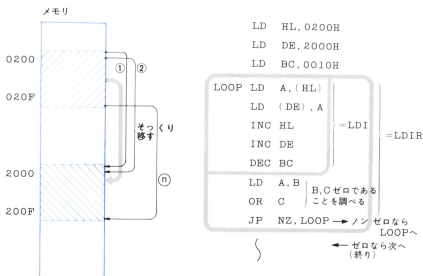
BC≠0のときはVフラグ，A−(HL)=0のときはZフラグがセットされる

ブロック入出力

命　令	動		作	注
INI	Cの内容を番地とするポートからHLの内容を番地とするメモリへ読み込む	HL←HL+1	1バイト入力して終る	Cは変わらない カウンタはBだけ（256まで）
INIR		B←B−1	B=0までくり返す	
IND		HL←HL−1	1バイト入力して終る	
INDR		B←B−1	B=0までくり返す	
OUTI	HLの内容を番地とするメモリの内容をCの内容を番地とするポートへ出力する	HL←HL+1	1バイト出力して終る	
OTIR		B←B−1	B=0までくり返す	
OUTD		HL←HL−1	1バイト出力して終る	
OTDR		B←B−1	B=0までくり返す	

B=0のときはZフラグがセットされる

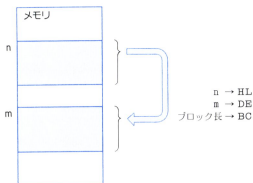
メモリ
0200
020F
2000
200F
①
②
そっくり
移す
ⓝ
LD HL, 0200H
LD DE, 2000H
LD BC, 0010H
LOOP LD A, (HL)
LD (DE), A
INC HL
INC DE
DEC BC
=LDI
=LDIR
LD A, B
OR C
B, Cゼロである
ことを調べる
JP NZ, LOOP
ノンゼロなら
LOOPへ
ゼロなら次へ
(終り)

メモリ
ⓝ
②
①
転送前と後のエリアが
重なっていると，最初のほうから転送した場合こ
れから転送すべき内容の所へ書いてしまい，内容
が正しく残らない．
このようなときは最後のほう（下）から転送する
＝LDDR 命令を使う

ブロック転送命令を使ったプログラム例

（ブロック転送サブルーチン）

HL レジスタに転送前アドレス，DE レジスタに転送後アドレス，BC レジスタに転送するブロックのバイト数が入っているとする．転送前と転送後のエリアが重なっている場合を想定して，正しく転送できるように〔LDIR〕と〔LDDR〕を使い分ける．

```
MVE  PUSH  HL       ; HL を退避
     AND   A        ; キャリをゼロに
     SBC   HL, DE   ; キャリを変化
     POP   HL       ; HL を元に戻す
     JR    NC, IRR  ; 上から転送
     DEC   BC       ; BC－1
     ADD   HL, BC   ; HL を変更
     EX    DE, HL
     ADD   HL, BC   ; DE を変更
     EX    DE, HL
     INC   BC       ; BC＋1
     LDDR           ; 下から転送
     RET
IRR  LDIR           ; 上から転送
     RET
```

54

ジャンプ命令

メモリに入れられたプログラムはゼロ番地から順次実行しますが，順序を入れ換えたり，条件によって別の番地のプログラムへ分岐したりするときに使います．

ジャンプ〔JP〕命令は，条件を伴わない無条件ジャンプと，条件付きジャンプ命令があり，飛び先番地は絶対番地が付けられます．

条件付きジャンプ命令は，フラグレジスタの内容によって，ジャンプするかジャンプせずに次の命令へいくかの決定をします．この条件には，ゼロフラグ，キャリフラグ，パリティフラグ，サインフラグがそれぞれ“0”である，または“1”であるとき，ジャンプせよというものです．条件に合わないときは，そのジャンプ命令がないのと同様に次の命令を実行します．

ジャンプリラティブ〔JR〕命令は，ジャンプ先のアドレスを絶対値で持つのではなく，現在のプログラムカウンタ（自分自身のアドレス）からの隔たりで持っています．数値としては −128 から +127 バイトで，0 のときは次の命令の先頭番地です．アセンブリ言語で書くときは，飛び先番地のラベルを指定すれば，自動的に計算してくれます．ジャンプリラティブ〔JR〕命令は 2 バイト命令ですが，ジャンプ〔JP〕命令（3 バイト）より実行時間がかかります．また条件判定は，ゼロフラグとキャリフラグについてのみしかできません．しかしジャンプ先までの隔たりが変わらなければジャンプ先がどこの番地に変わってもマシン語は同じになりますので，小さなプログラムモジュール内で使った場合，モジュールの配置は自由（リロケータブル）となります．ディクリメントジャンプノンゼロ〔DJNZ〕命令は，変わった命令で，B レジスタから 1 を引き，ゼロでなければジャンプリラティブと同じ方式で示された番地へジャンプするというものです．くり返しに入る前に B レジスタへ必要回数を書き，くり返し処理の終わりにこの命令をおけば，終了しなければもう 1 回くり返しの先頭へジャンプ，終了すれば抜け出て次へ，と分岐させることができます．回数は B レジスタが 8 ビットですから 256 回（B レジスタにゼロを入れておくと 256 回で終わる）までです．

無条件ジャンプ

命　令	オ ペ ラ ン ド
JP　nn	飛び先のメモリにつけられた番地をそのままオペランドにつける（**絶対番地**） （例）JP 0258H ⟹ マシン語では C3 58 02（下位 上位） アセンブリ言語ではラベル名を書いてもよい オペランドには（HL）（IX）（IY）を使用してもよい．
JR　nn	飛び先のメモリの番地までのディスプレイスメントをオペランドにつける（**相対番地**） （例）JR LO2 ⟹ LO2 が0255H番地で次の命令が0250H番地 とすると，マシン語では18 05

条件付きジャンプ　　条件が合えばジャンプする．合わなければ次の命令へ

命令		条　　件
JP	JR	
JP Z, nn	JR Z, nn	Z フラグを調べゼロの状態（Z フラグ＝1）ならジャンプ
JP NZ, nn	JR NZ, nn	Z フラグを調べノンゼロの状態（Z フラグ＝0）ならジャンプ
JP C, nn	JR C, nn	C フラグを調べキャリ有り（C フラグ＝1）ならジャンプ
JP NC, nn	JR NC, nn	C フラグを調べキャリなし（C フラグ＝0）ならジャンプ
JP PE, nn	—	P/V フラグを調べイーブン（偶数）（P/V フラグ＝1）ならジャンプ
JP PO, nn	—	P/V フラグを調べオッド（奇数）（P/V フラグ＝0）ならジャンプ
JP M, nn	—	S フラグを調べマイナス（S フラグ＝1）ならジャンプ
JP P, nn	—	S フラグを調べプラス（S フラグ＝0）ならジャンプ
—	DJNZ nn	B←B−1　B≠0ならジャンプ

注　JRのオペランドはアセンブラによってはラベル名ではなく，相対値を書くものもある．

```
        ⌇
        LD    B,10H
        LD    HL,2000H
        LD    A,00H
LO1     ADD   A,(HL)
        INC   HL
        DJNZ  LO1        ─→ Bレジスタから1を減じゼロなら次の命令を実行する.
                            ゼロでなければLO1番地へジャンプ
        JP    KEYR       ─→ KEYRとラベルをつけた番地へジャンプ
                            (この次はKEYR番地の命令を実行する)
CRTR    LD    A,30H      ←─ ここへは他にあるジャンプ命令等でジャンプしてくる.
        LD    C,00H
        ⌇
        JR    LO2        ─→ (LO2のラベルがつけられた命令の先頭番地－次の命令の先
KEYR    IN    A,(08H)       頭番地)の値をマシン語のオペランドに持つジャンプ命令(2
                            バイト構成の命令).LO2とラベルをつけた番地へジャンプ
        CP    0
        JP    Z,KEYO     ─→ 前の命令でAレジスタと0を比較してゼロ(一致)ならKEYO
                            番地へジャンプ
        CP    1          ←─ ゼロでなければこの命令を実行する
        ⌇
KEYO    LD    B,20H
LO2     LD    HL,0038H
        ⌇
```

よいプログラムとは

1．実行速度が速いこと

用途によっては，速度が問題にならない場合もあるが，まずほとんどの場合，全システムの効率に大きく影響する．

2．メモリをくわないこと

プログラム自体の長さは，無駄を省いた最小限でなければならない．

実行速度を上げるためにも重要である．また，専用システムではメモリ数を減らし，部品点数を引き下げることにより得られるメリットは多い．ただし，以下の項目を犠牲にしないよう心がけなければならない．

データ格納用メモリは，情報処理（事務計算）のような用途で多用されるが，フロッピディスクなどの外部記憶を有効に利用し，システム効率を上げるような設計が望ましい．

3．デバッグ（修正）しやすいこと

いかに有能なプログラマでも，できあがったプログラムが1回で動作するのはまれである．また，できあがってから何年もたってからバグ（誤り）が，発見されることがある．すみやかに対処できるよう配慮しなければならない．次項とも共通するが，コメントを豊富に，機能をモジュール化し分散させること，フラグの使用は最小限にする．サブルーチンからジャンプでメインルーチンへ戻るなど論外である．

4．改造しやすいこと

新しくプログラムを作るとき，前に作ったプログラムの改造で対応できれば，こんな楽なことはない．設計変更されそうな点を予測すること．

5．速く作ること

プログラムの原価は，ほとんど人件費につきる．速く，しかも後で応用のきくプログラムを作ることが，プログラマの使命である．

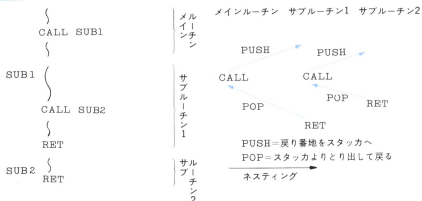
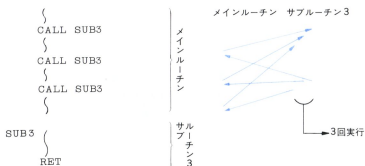
55 コール，リスタート，リターン命令（サブルーチン）

コール〔CALL〕**命令**は，ジャンプ命令と似ていますが，ジャンプするとき次の命令の番地をスタッカへ自動的にプッシュ〔PUSH〕します．そしてリターン〔RET〕命令があると，スタッカからポップ〔POP〕した値をプログラムカウンタへ入れて元の流れへ戻ります．コールで呼び出される番地からリターン命令までを**サブルーチン**と呼び，プログラムの中のどこからでも呼び出して実行させることができます．コール命令は，2バイトのジャンプ先の絶対番地を持った3バイト命令です．コール命令にも，ジャンプ命令と同じ条件を付けることができます．条件を満たしていないときは，コール命令がないのと同様に，ジャンプせずに次の命令を実行します．

リスタート〔RST〕**命令**は，特定の8個の番地へのコール命令に相当します．1バイト命令なので，多用されるサブルーチンをコールするときに便利な命令です．

リターン命令であっても，ノンマスカブル割り込み〔NMI〕からのリターンはリターンフロムノンマスカブルインタラプト〔RETN〕を使います．他の割り込みに対する割り込み禁止状態を解除するためです．また，割り込み（INT）からのリターンは，Z-80のペリフェラルのデージーチェーンによる優先順位決定機能を使うときは，リターンフロムインタラプト〔RETI〕命令を使って，ペリフェラルに対して割り込み処理の完了を知らせてやる必要があります．これは割り込み処理ルーチンの中で，サブルーチンコールを行なう場合もあり，サブルーチンからのリターンで割り込み処理完了になってしまわないようにするためです．

サブルーチンの使い方は，プログラムの良否にかかわる重要なテクニックです．単に同じ手続きをあちこちで何回も使うから，記述の手間を省くというだけでなく，プログラムの機能のブロック化，あるいは単機能モジュールのブラックボックス化に役立ちます．デバッグ済みのサブルーチンは内容を知らなくても使い方だけを知っておけば，いつでも呼び出せます．モジュールとしてたくさん作っておけば，次からはモジュールの組み合わせを変えるだけで，新しいプログラムができあがるわけです．

例 1 メインルーチンからサブルーチン1をコールし，さらにサブルーチン2をコールする

例 2 メインルーチンのあちこちから1つのサブルーチン3をコールする

命　令	オ　ペ　ラ　ン　ド	
CALL nn	nnは絶対番地	無条件コール
CALL cc，nn	nnは絶対番地 ccは条件	条件付コール　前項のJPと同じ条件
RET	オペランドはない	無条件リターン
RET cc	ccは条件	条件付リターン　前項のJPと同じ条件
RST n	nは 00H 08H 10H 18H 20H 28H 30H 38H のいずれかを選ぶ n=08Hとすると，0008Hへのコール命令（CALL 0008H） と同じ働きをする．ただしRST nは　1バイト命令	

56

Fレジスタとフラグ変化

Fレジスタは六つのフラグにより構成されます．フラグはそれぞれが意味を持ち，演算命令やローテート命令，入出力命令などの実行結果によって変化します．

キャリフラグ（Cフラグ）は，最上位のビットからの桁上がり，桁借りにより，あるいはローテート，シフト命令で変化します．

ゼロフラグ（Zフラグ）は，実行結果がゼロになったとき“1”にセットされます．

パリティ/オーバフローフラグ（P/Vフラグ）は，パリティ（ビットパターンの“1”の数が奇数か偶数か）と，符号付き演算のオーバフローを兼ねています．パリティは，奇数なら“0”，偶数なら“1”，オーバフローは，オーバフローして本来正の数になるべき結果が負の数になってしまったとき“1”，正しい結果のとき“0”になります．

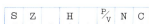
サインフラグ（Sフラグ）は，符号付き演算の符号ビット（左端ビット）と同じになります．負の数であれば“1”，正の数であれば“0”です．

ハーフキャリフラグ（Hフラグ）は，下位4ビットに対するキャリフラグです．

これらのフラグのうち，サブトラクトフラグとハーフキャリフラグは，ジャンプ命令では判定できませんが，2進化10進数を扱うデシマルアジャストアキュムレータ〔DAA〕命令を実行するとき参照されます．

フラグは命令により変化する，変化しない，不定になる，各場合があります．

コンプリメントキャリフラグ〔CCF〕命令は，他のレジスタに変化を与えずにキャリフラグのビットパターンを反転させる命令です．セットキャリフラグ〔SCF〕命令はキャリフラグを“1”にする命令です．なお“0”にしたいときは〔AND A〕か〔OR A〕を使います．

どの命令を実行したときどのフラグが変化するかは，巻末のフラグ変化表と命令表を参照してください．

Fレジスタ
S Z H P/V N C
キャリフラグ
加算，減算で桁上り，桁借りがあったとき1になる．
ローテート，シフトでは，各ビットにつれて移動する．
サブトラクトフラグ
（DAA命令実行用＝プログラムでは操作できない）
命令によりセットされるかリセットされるか決まっている．
加算系のとき0，減算系のとき1になる．
P/Vフラグ
命令によりP：パリティフラグとV：オーバフローフラグに使いわける．
パリティフラグ
論理演算，Cレジスタ間接ポート指定の入力命令等で，ディスティネーションに乗ったデータのビットパターンの1の数が偶数（イーブン）のとき1，奇数（オッド）のとき0になる．
オーバフローフラグ
演算の結果が符号付算術演算の結果としてオーバフローしたとき1．すなわち8ビット演算では−128〜＋127，16ビット演算では−32768〜＋32767以外を表現する必要が出たときセットされる．したがって，正数のみを対象とした符号なし演算，(8ビットでは0〜255，16ビットでは0〜65535の範囲で定義した演算)のときは無視してよい．この場合はCフラグがオーバフローを表わしている．
このビットは使用していない
ハーフキャリフラグ
（DAA命令実行用＝プログラムでは操作できない）
8ビットの演算で下4ビットからの桁上げ，桁借りがあったとき1になる．
このビットは使用していない．
ゼロフラグ
実行の結果がゼロのとき1，ゼロ以外のとき0になる．
サインフラグ
実行の結果が符号付算術演算の結果として負の数になったとき（この場合最上位ビットが1になる）1になる．

57

2進化10進数と10進補正命令

CPU内の算術演算は，メモリやレジスタに記憶されたビットパターンを2進数として演算します．8ビットレジスタでは0～255まで，256以上を表わすと桁上げをしますが，レジスタの内容は0と256が同じパターンになります．多くの桁を扱う場合は，桁上げ操作や入出力を簡単にするため，2進化10進数（バイナリコーデッドデシマル＝BCD）を用います．

4ビットのパターンは16種類0～Fまでありますが，このうち0～9までを有効とし，A～Fについては上位桁の4ビットへ1くり上げ0～5にします．このようにすると4ビット＝1桁の10進数表現ができます．10桁の10進数を使いたいときはメモリに5バイトのエリアを確保し，2桁ずつ演算するのです．

しかし，演算はBCDで表現されていても，2進数で計算しますので，結果としてA～Fが出現してしまいます．ここでデシマルアジャストアキュムレータ〔DAA〕命令を実行すると，いまの計算が加算か減算か，ハーフキャリは出ているか，などフラグレジスタの状態によって必要な桁上げ，桁借り処理をして，BCDに戻してくれます．BCDはデータエリアが多少大きくなりますが，入出力はほとんどの場合10進数が要求されますので，変換の手間が省け，多桁演算がやりやすいなど特徴が多く，よく使われる手法です．

ビットパターン	名前	BCD 表現
0 0 0 0	0	ここまでを 10進数に対応 させる 1010 は 上の位へ桁上げし 0000 に戻る
0 0 0 1	1	
0 0 1 0	2	
0 0 1 1	3	
0 1 0 0	4	
0 1 0 1	5	
0 1 1 0	6	
0 1 1 1	7	
1 0 0 0	8	
1 0 0 1	9	
1 0 1 0	A	使わない
1 0 1 1	B	
1 1 0 0	C	
1 1 0 1	D	
1 1 1 0	E	
1 1 1 1	F	

8＋6 を例にとって
加算すると
10進数では
8＋6＝14 であるが
2桁の BCD では

```
   0000 1000 = 08
+) 0000 0110 = 06
   0000 1110 = 0E
```

となり，下4ビットは使わない値に
なってしまう．
ここで DAA 命令を実行すると

0000 1110
↓
0001 0100
↓　　↓
1　　4

になおしてくれる
この補正のし方は加減算，
桁あふれ等により異なるが，
F レジスタの働きで間違えることはない

BCD の表し方

1001　0110
9　　6
↑　　↑
10の位　1の位

2桁の10進数

（4ビットを1デジット（桁）と呼ぶのは
ここからきている）

2進数

1001　0110　は

16進数 になおすと（名前で呼ぶと）

96 である．これを
10進数になおすと
$9 \times 16^1 + 6 \times 16^0 = 150$　である

これを BCD と定義すると

1001　0110　は

16進読み にすると

96（きゅうろく）
10進数にしても
96（きゅうじゅうろく）である

コンピュータの中で数値を表わすには

1. 2進数として扱う
2. BCD として扱う
3. 指数表現をとり入れる

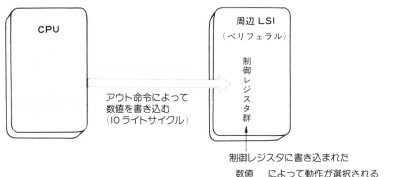
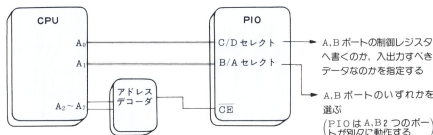
等のやり方がある

- 1．は入出力のたびに10進数との変換が必要．プログラムは簡単でスピードも早い．桁数が自由にならない．
- 2．は桁数が自由になり，入出力，プログラムは簡単．メモリにむだが多い．
- 3．は，桁数が多い場合に有効．プログラムは複雑で演算時間がかかる．

58 ペリフェラルのプログラミング

ペリフェラルを使うときは，データの入出力を行なう前に，ペリフェラルの内部レジスタへ情報を書き込んで，目的にあった動作をするように設定しなければなりません．機能が多いほど選択の幅が広くなりますので，たくさんの情報を与えなければなりません．情報は次々に書き込みます．情報の中に何の情報かを示すビットが設けられている場合もありますが，書き込む順序によって，意味が違ってしまうことがあります．マニュアルに記載された順序を守れば安全です．

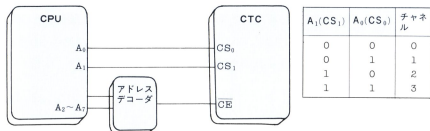
ペリフェラルのレジスタへ情報を書き込むには，そのペリフェラルのつながっているIOポートへ出力命令を出します．PIOとSIOには，C/Dセレクト信号端子があり，“1”のときCすなわち制御語（control word），“0”のときDすなわちデータと解釈しますので，制御情報を書き込むときはこの端子を“1”にして出力命令を出します．実際にはこの端子をアドレスバスのいずれかのビットへつなぎますと，制御語とデータとは別のIOポートアドレスに配置されることになります．CTCのチャネルセレクトやPIOのB/Aポートセレクトも同じ考え方で対処します．

一度書き込んだ制御情報は，リセットされるか，または新たに書き込まれるまで有効です．また，特別な場合を除いてIN命令で読み出すことはできません．

PIO の場合

$A_7 \sim A_2$	A_1	A_0	番地	ポート切換え	データ/制御語切換え
×	0	0	×0，×4，×8，×C のどれか	B/A セレクトが "0" だから A ポート	C/D セレクトが "0" だからデータ
×	0	1	×1，×5，×9，×D のどれか		〃　"1"　〃　制御語
×	1	0	×2，×6，×A，×E のどれか	B/A セレクトが "1" だから B ポート	〃　"0"　〃　データ
×	1	1	×3，×7，×B，×F のどれか		〃　"1"　〃　制御語

×はアドレスデコーダによってきまる

CTC の場合　（CTC に対する入出力は制御レジスタにのみ適応する　CS_0 CS_1 は内部の 4 つのチャネルを選択する）

$A_1(CS_1)$	$A_0(CS_0)$	チャネル
0	0	0
0	1	1
1	0	2
1	1	3

59

PIO モード 0 の動作

モードOは出力モードです．8ビットの入出力線に出力〔OUT〕命令で，出されたデータバスの状態がそのまま出てきます．データバスの内容が変わっても，次に出力命令がくるまで出力線の内容はそのままです．

CPUからの出力動作では，IOリクエスト（$\overline{\text{IORQ}}$）とライト（$\overline{\text{WR}}$）信号が“L”になりますが，PIOにはライト信号の入力端子がありません．これは（$\overline{\text{IORQ}}$）が“L”で，リード（$\overline{\text{RD}}$）が“H”で，チップイネーブル（$\overline{\text{CE}}$）が“L”で，$\overline{\text{C/D}}$がデータすなわち“L”のとき，PIO内部でライト信号を作り出しているからです．

出力命令が出されると，データバスからPIOの出力レジスタへ信号が入ります．内部のライト信号が立ち上がると次のクロックの立ち下がりで，すでに出力線に出力レジスタの内容が乗り確定しています．同時にレディ信号線からレディ（RDY）信号が出されます．外部装置はこのレディ信号が“H”になったことにより出力線のデータを取り込み，終わったらストローブ（$\overline{\text{STB}}$）信号線を“L”にして応答します．レディ信号は，このストローブの立ち上がりの次のクロックの立ち下がりで“L”に戻り，次のデータの出力命令を待ちます．割り込みがイネーブルにプログラムされていて，優先順位の高い割り込みがかかっていなければ，ストローブの立ち上がりで割り込み（$\overline{\text{INT}}$）出力は“L”になってCPUに割り込みをかけることができます．次の出力命令があるまで出力線の状態は変わりません．

レディ信号が“H”のとき，すなわち出力命令に対してストローブが返っていないときにさらに出力命令を出すと，レディ信号は一度“L”に戻ってすぐ“H”になります．レディ信号は正論理信号ですから“H”アクティブです．

レディとストローブの信号線をハンドシェーク線と呼びます．外部装置とCPUはPIOのハンドシェーク線を通じて，互いに確認し合いながら（同期をとりながら）データのやりとりを行なうわけです．

CPU
OUT 命令
INT
❶ PIOにデータを送る
IOライトサイクルに従って送り出す
❻ 割り込み
PIOから外部へデータが出ていったことをCPUに知らせる
割り込みを発生させなくすることもできる
PIO
RDY
ポートデータバス
STB
❷ データが来たよ
(レディ)
データ
❸ データを受けとる
❹ データを受け取ったよ(ストローブ)
❺ ストローブが戻ったところでレディを元に戻す
外部装置
レディが来たらデータを受け取り，ストローブを返すように作る
クロック
(CPUのサイクル)
(T_2)
(T_W)
(T_3)
OUT命令
(IOライトサイクル)
当該PIOポートに対するWR信号
ポート出力
前のデータ
CPUからのデータ
(OUT命令によって書き込まれた)
レディ(RDY)
ストローブ
(STB)
外部から入れる
INT
ストローブの立ち上りの次のクロックの立ち下りでレディを元に戻す
CPUの割り込み応答サイクルへ
データバスのデータを取り込む
IOライトサイクルの終了の次のクロックの立ち上りでレディを“H”に
取り込んだデータをポートデータバスへ出力する
ストローブの立ち上りでINTを“L”に

60

PIO モード1の動作

モード1は入力モードです．外部から入ってきた8ビットの信号がポートの入出力線に与えられ，CPU からの入力〔IN〕命令でデータバスを通じてCPU の内部レジスタへ取り込みます．

レディ端子は，リセットがかかると“L”になっています．CPU からの入力命令で，“H”になって受け入れ可能を外部へ知らせます．この1発目の入力命令で読み込んだデータは無意味です．外部装置からレディ信号線が“H”のとき，入出力線にデータを与え，続いてストローブを“L”にするとストローブの立ち上がりで，PIO は入出力線から内部の入力レジスタへデータを取り込みます．次のクロックの立ち下がりで，レディを“L”にして外部から次のデータが入ることを禁止し，同時に割り込み可の状態であれば，CPU に割り込みをかけます．割り込み処理ルーチンで，入力〔IN〕命令を実行し，このデータを CPU の内部レジスタへ読み込みますと，リード（$\overline{RD}$）信号の立ち上がりと次のクロックの立ち下がりでレディ信号線を“H”に戻し，次のデータ入力に備えます．

モード1では1個のダミーの入力命令が必要です．プログラミングが終わり，外部からの信号を受け入れてもよい状態になったら〔IN〕命令を実行して，レディ信号を出し外部装置に受け入れ可を知らせてください．レディ信号が受け入れ状態を示していないときに，入力線にデータを乗せてストローブ信号を与えても，入力レジスタ上へデータを書き込むことはできます．しかしなんらかの方法で CPU の入力命令の実行が遅れないようにしないと，データが失われるおそれがあります．

CPU
IN命令
INT
❶ データを読み込む
IOリードサイクルに従って読み込む
❼ 次のデータを読み込む
❻ 割り込み
外部からPIOにデータが入ったことをCPUに知らせる
割り込みを発生させなくすることもできる
PIO
RDY
STB
❷ CPUがデータを読み終ったから次を送ってもいいよ
(レディ)
❸ データを送る
❹ データを送ったよ
(ストローブ)
❺ ストローブが入ったところでレディを戻し，CPUが読み込むまで次のデータを送らないよう外部に待ったをかける
外部装置
レディがきたらデータを送り，ストローブを返すように作る
クロック
ストローブ
(STB)
ポート
データバス
何でもいい
データ
INT
割り込み応答
サイクルへ
ストローブ立ち上り後次のクロックの立ち下りでレディが落ちる
RDY
ポートデータバスのデータを取り込む
(T2) (Tw) (T3)
CPUのIOリードサイクル
IN命令 (IOリードサイクル)
当該PIOポートに対するRD信号
RDの立ち上りの次のクロックの立ち下りでRDYは戻る

61

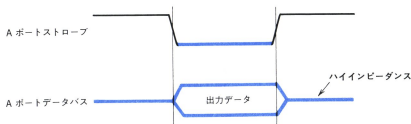
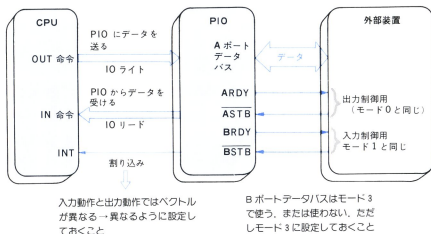
PIO モード2の動作

モード2はAポードだけについて可能な，入出力モードです．8本のデータ線は入出力共用になります．ハンドシェーク線はAポートが出力制御用，Bポートが入力制御用になります．したがって，Bポートはハンドシェーク線を必要としないモード3に設定しなければなりません．

出力命令によるデータ出力動作はモード0と同じですが，モード0では出力線に常にデータが乗せられているのに対し，モード2のときはAポートのストローブ（$\overline{\text{ASTB}}$）信号線が“L”になっている間だけ，多少遅れて乗せられてきます．外部装置はこのストローブを立ち上げるタイミングで，データを読み取ればよいのです．入力命令によるデータ入力動作は，Bポートのハンドシェーク線を使うほかはモード1と同じです．

割り込みをかけるタイミングも，モード0，モード1と同じですが，入力動作ではBポートのハンドシェーク線を使うため，Bポートに書き込まれたベクトルが送出されます．したがって，入力，出力各動作で割り込みを別々のベクトルで制御することができますが，Bポートをビットモードで使用する場合は，本来のBポートの動作で発生する割り込みに対するベクトルと，Aポートの入力動作で発生するベクトルが同じ値になってしまいます．いずれかの割り込み機能を使用しないようにするか，処理プログラムの中で，どちらの割り込みかを判断しなければなりません．

入力動作と出力動作ではベクトルが異なる→異なるように設定しておくこと

Bポートデータバスはモード3で使う．または使わない．ただしモード3に設定しておくこと

Aポートストローブが“L”のときだけ出力データがポートデータバスに現われる．
したがってこのときは外部からデータを与えてはならない．
他は，モード0，モード1と同じ．

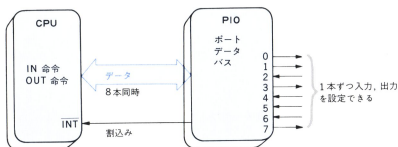
62

PIOモード3の動作

モード3はハンドシェーク線を使用しないで，非同期で入力出力を行います．ビット単位に入出力の設定ができ，割り込みは入力に指定したビットが指定した状態になったときに入ります．

出力命令で送られたデータは，モード0と同じタイミングで出力に指定されたビットに乗せられます．入力命令で読み込むと，リード（$\overline{RD}$）信号が立ち下がる直前の入力に指定されたビットの状態がCPUに読み込まれます．出力に指定されたビットは，それ以前に出力命令で送り出したままが読み込まれます．

入力に指定したビットは，さらに割り込みに関係させるか否かのマスク指定をすることができます．マスク指定で割り込み要因のモニタビットに指定したビットが全部そろったとき（AND条件），いずれか一つが入ったとき（OR条件）を選択でき，入力線を正論理（“H” アクティブ）としてとらえるか，負論理（“L” アクティブ）としてとらえるかの選択もできます．

コントローラとしての用途では，PIOをモード3で使うことが多いと考えられます．周期的にCPUは入力ビットの状態を検査し，対応した出力信号を出力ビットへ乗せるようなプログラムを組みますが，入力ビットを読み込む周期はプログラムの長さで決まります．もっと早い対応が必要なシステムでは，割り込みを使うことにより解決できます．ただし，割り込みによるシステムの応答は，外部装置で起こり得る最悪の条件のときを想定して設計しないと，CPUが追いつかなくなることもあります．時間計算から許容ステップ数を出し，その範囲内のプログラムを作るのですが，システムクロック周波数の決定とも合わせて慎重に検討しなければなりません．

割り込み発生の条件，設定

1．入力ビットのうちどれを判断の対象とするか（マスク）
2．入力は "H" アクティブか "L" アクティブか　（論理）
3．AND か OR か　(AND/OR)

たとえば，上の例で

・ビット 4 と 6 が共にアクティブになったとき割り込みを発生させたい

・入力は普通は "H" になって信号があるときだけ "L" になる（"L" アクティブ）

とすると

・マスク指定はビット 4 と 6

・論理は負論理

・AND/OR は AND

と指定する

具体的には（次頁参照）

ベクトル	××××××××0	⇒	××
モード設定	11001111	⇒	CF
ビット指定	01110100 ↑7ビット目　↑0ビット目	⇒	74
割込み制御語	11010111	⇒	D7
マスク指定	10101111	⇒	AF

⇩

これを順次PIOの
制御レジスタ群へ
OUTする

63

PIOのプログラミング

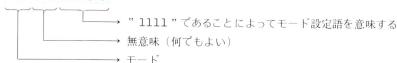
割り込みベクトル

7 6 5 4 3 2 1 0

- "0" であることにより割り込みベクトルを意味する
- 割り込み処理ルーチンのアドレステーブルの番地（下位）

モード設定

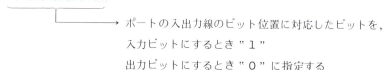
7 6 5 4 3 2 1 0

- "1111" であることによってモード設定語を意味する
- 無意味（何でもよい）
- モード

7 6

00＝モード0（出力モード）
01＝モード1（入力モード）
10＝モード2（双方向モード）
11＝モード3（ビットモード）

ビット指定　モード設定でモード3を指定したときは続けてこの指定をします．

7 6 5 4 3 2 1 0

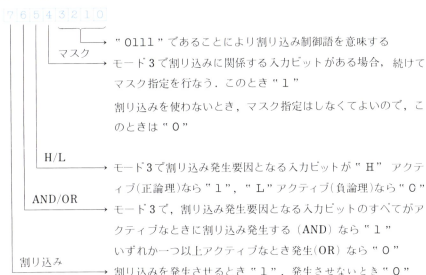
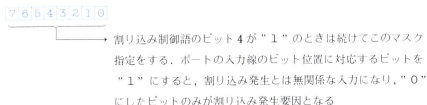
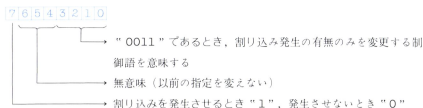
- ポートの入出力線のビット位置に対応したビットを，
入力ビットにするとき "1"
出力ビットにするとき "0" に指定する

> 制御語の書き込みはAポート，Bポートを別々に行なってください．
> A，B混在させると，動作しない場合があります．順序もここに書かれているとおりなら問題ありません．
> 割り込みを使わない場合は，ベクトルの書き込みは省略できます．

割り込み制御語

マスク指定

割り込み制御語のビット4が"1"のときは続けてこのマスク指定をする．ポートの入力線のビット位置に対応するビットを"1"にすると，割り込み発生とは無関係な入力になり，"0"にしたビットのみが割り込み発生要因となる

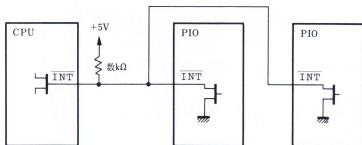
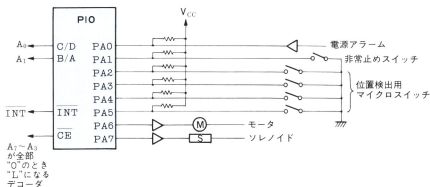
割り込み制御語

64

PIOのプログラム例

PIOをモード3で使用する場合の一例を考えてみます．PIOはIOポートの00H～03Hに配置されているものとします．Aポートの0～5ビットを入力とし，他を出力ビットにします．電源異常信号と非常停止信号が共に“L”アクティブで，0と1ビットにつながっています．処理に速度が要求されますので，割り込みをモード2で使用します．他の入力は位置検出のマイクロスイッチの信号で，出力はすべてモータやソレノイドを駆動します．出力を“H”にすると動き，“H”にすると止まります．応答速度は問題にならないとします．

まず0000H番地からのルーチンでPIOを設定します．設定の最後でCPUの割り込み受け付けを可能にします．次に入力ビットを読み込み，その条件に従った制御情報をAレジスタにととのえ，出力ビットへ出力します．

割り込みが入ると，すべての出力を“L”にして動作を止めます．CPUはホルトに入り，新たにリセットがかかるのを待ちます．

PIOに限らず，ペリフェラルの $\overline{\mathrm{INT}}$ の出力端子はオープンドレインになっています．したがって，＋5Vへ抵抗でプルアップしておかなければなりません（ワイアードオア）．

```
;   CONTROL                     コメント行
        ORG     0000H           ;このプログラムをゼロ番地から配置
        LD      SP, 0000H       ;スタックポインタイニシャライズ
        IM      2               ;割り込みモード
        LD      A, 01H          ;Iレジスタセット
        LD      I, A
        LD      A, 00H          ;ベクトル
        OUT     (01H), A
        LD      A, 0CFH         ;PIOモード設定          PIOのプログラミング
        OUT     (01H), A
        LD      A, 3FH          ;ビット指定
        OUT     (01H), A
        LD      A, 97H          ;割り込み制御語
        OUT     (01H), A
        LD      A, 0FCH         ;マスク指定
        OUT     (01H), A
        EI                      ;割り込み許可
JOBS    ∫                       ;仕事を始める

        IN      A, (00H)        ;入力ビットを読む
        ∫
        OUT     (00H), A        ;出力をコントロールする
        JP      JOBS            ;次の仕事を始める

        ORG     0100H
        DEFW    INTR            ;割り込み処理ルーチンの先頭番地定義
;                               ;割り込み処理ルーチン
INTR    LD      A, 0H           ;モータを止める
        OUT     (00H), A
        HALT                    ;CPUを止める．リセット信号が入るまで停止
        END                     ;プログラム終り
```

65 CTC カウンタモード

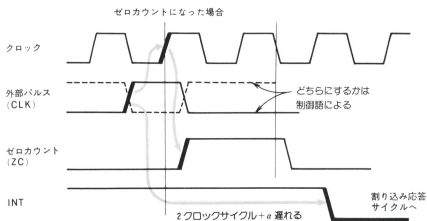
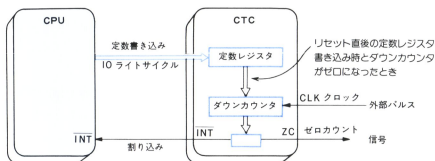
CTC のクロック/トリガ (CLK/TRG) 端子に，与えられるパルスをカウントします．あらかじめ設定された数を 1 個のパルスで 1 ずつ減らしてゼロになると，次のクロックの立ち上がりでゼロカウント/タイムアウト (ZC/TO) 端子を 1.5 クロックの間 "H" にし，割り込みを発生します．

カウントするパルスはシステムクロックの 2 倍以上の周期で，"H" と "L" の時間は，おのおの 120ns 以上必要です．パルスの立ち上がりまたは立ち下がりエッジ(指定できる)がくると，次のクロックの立ち上がりで，ダウンカウンタを 1 減じます．ダウンカウンタがゼロになるとゼロカウント信号を出し割り込みをかけます．同時に定数レジスタに設定されている数値をダウンカウンタへ書き込み，次のカウントに備えます．カウント途中で定数レジスタに新しい数値を書き込むと，次のゼロカウント時から有効になります．定数レジスタからダウンカウンタへの書き込みは，リセット状態から最初に定数レジスタが設定されたときと，ダウンカウンタがゼロになったときです．定数レジスタは 8 ビットですから 1〜256 の値が設定できます．0 を設定したときは 256 を意味します．

割り込みベクトルレジスタは，CTC 1 個＝4 チャネル分に 1 個しかありません．ベクトルの設定はチャネル 0 に書き込みます．割り込みが発生すると要求するチャネルによって，ビット 1 とビット 2 が決まったパターンに修飾されて送り出されてきます．優先順位はチャネル 0 が一番高く，順にチャネル 3 が最後位です．

制御語や定数の書き込み，読み出しのときのチャネル指定は，チャネルセレクト (CS_0) と (CS_1) を切り換えて行ないます．アドレスバスの 2 本 (A_0 と A_1 がよく使われる)を接続すれば，CPU 側からは四つのチャネルが一連のポートアドレスに配列されます．PIO の C/D セレクト，B/A セレクトと同じ考え方です．

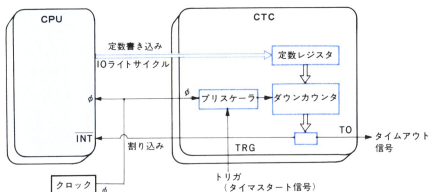
CPU
CTC
定数書き込み
IO ライトサイクル
定数レジスタ
リセット直後の定数レジスタ
書き込み時とダウンカウンタ
がゼロになったとき
ダウンカウンタ
CLK クロック
外部パルス
INT
INT
ZC
ゼロカウント
信号
割り込み
ゼロカウントになった場合
クロック
外部パルス
(CLK)
どちらにするかは
制御語による
ゼロカウント
(ZC)
INT
割り込み応答
サイクルへ
2クロックサイクル+α 遅れる

66

CTC タイマモード

カウンタモードでは，不定期的なパルスをカウントしますが，決まった周期を持ったパルスをカウントすることにより，時間経過を知ることができます．これがタイマモードの動作の基本です．カウントするパルスはプリスケーラで，1/16 か 1/256 に分周されたシステムクロックを使います．システムクロックの周期と分周数と，定数レジスタに書かれた数値の積がゼロカウント/タイムアウト（ZC/TO）端子へ信号が出てくる時間周期となります．

タイマの起動は，トリガと自動が選択できます．トリガ起動の場合は，クロック/トリガ端子の立ち上がりか立ち下がりのエッジ（選択できる）から 2 回目のクロックの立ち上がりでスタートします．自動の場合は，時間定数を書き込む出力命令の次の命令サイクルと同時にスタートします．

割り込みの発生と，定数の再設定についてはカウンタモードと同じです．

CTC はカウント中に，読み出しても差しつかえありません．ダウンカウンタの現在値を知ることができます．ゼロカウント/タイムアウトを検知するのに ZC/TO 信号も割り込みも使わず，ダウンカウンタの値を読み出し，ゼロかどうか調べる方法が考えられますが，ダウンカウンタがゼロになり，次に定数レジスタの値が再びダウンカウンタへ書き込まれるまで，2 クロックしかないので，この間に CPU が読み出すとは限らないため，不確実で実用できません．

前項では φ は省略してあるが,
カウンタモードでもクロックは
与えなければならない

トリガ起動の場合

クロック

トリガ
(TRG)

タイマスタート

タイムアウトの場合

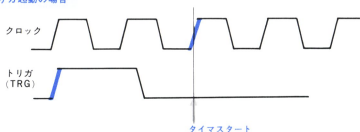
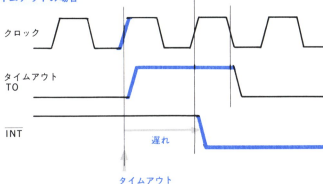
クロック

タイムアウト
TO

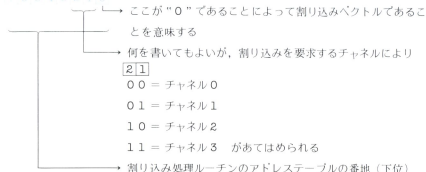
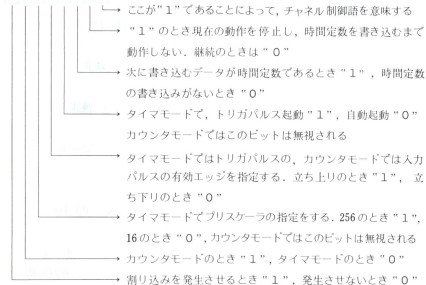
INT

遅れ

タイムアウト

67

CTCのプログラミング

割り込みベクトル

（チャネル0に対してのみ書き込むことができます）

チャネル制御語（各チャネルごとに書き込みます）

時 間 定 数

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

→ チャネル制御語のビット2が“1”のとき続けて書き込む．各ビットは2進数でゼロカウントになるまでの数値を表現している．ゼロのときは256として扱われる

8255との接続

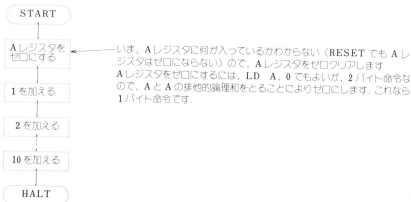
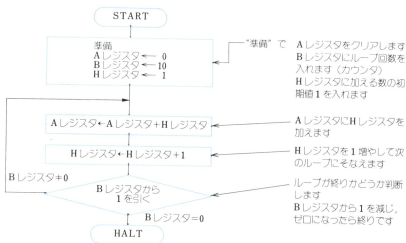
Z-80ファミリには，パラレル入出力ポートとしてPIOが用意されて，ハンドシェークや割り込み機能など使いやすい機能が盛り込まれています．しかしメカニズムの制御で使われる入出力は，単純なビット単位のオンオフが多く，このような用途の場合はいろいろな機能よりむしろ，1本でも多くのポートを持っていることが選択の条件になります．インテル社の8255は，PIOのような割り込みの機能は持っていませんが，8ビットポートが3本と，PIOより多くのポートを持っており，上のような用途によく使われています．このような応用には，ICの数が増えるのをいとわなければ，標準ロジックICのD-フリップフロップなどを使っても構成できます．

8255はZ-80ファミリとは設計メーカも違い，1世代前の製品ということもあって，完全な結合はできません．しかし機能を限定することによって，十分目的を達することが可能です．

例　　　題

プログラム 1 （ループ）
プログラム 2 （判　断）
プログラム 3 （メモリクリアサブルーチン）
プログラム 4 （変換-テーブルサーチ）
プログラム 5 （スイッチの表示）
プログラム 6 （スイッチの表示-割り込み）

プログラム 1 （ループ）

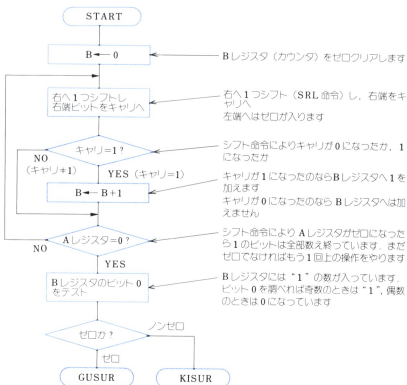
1 から 10 までの整数を加えるプログラムを考えます．答は A レジスタに入っていればよいとします．〈SAMPLE 1〉

これが 1～100 までだったら上のやり方では大変です．そこで「くり返えし」を使います．〈SAMPLE 2〉

この方法なら，B レジスタに入れる数を変えればいくつでも加えられます．ただし，このままではA レジスタがオーバフローすることがありますので工夫がいります．

〔注〕 サンプルプログラムを全部一連にしてアセンブルしたものです．各サンプルプログラム間に関連はありません．実行するときにはモニタプログラムを持ったワンボードコンピュータ．たとえばシャープのSM-B-80TEを使ってください．

```
行番号                                ソースプログラム（キーを打ってインプットする）
 1  プログラムを      ;****************************************
 2  見やすくする      ;SAMPLE1
 3  ためのコメン      ;
 4  ト行              ;
    アドレス マシン語  ラベル  オペコード オペランド
 5                           ORG    0000H <--- このプログラムを0000H番地から配置することをアセンブラに知らせるアセンブラ命令
 6  0000 AF                  XOR    A     <--- Aをゼロにする
 7  0001 C601                ADD    A,1        A←A+1  Aは1になる
 8  0003 C602                ADD    A,2        A←A+2  Aは3になる
 9  0005 C603                ADD    A,3        A←A+3  Aは6になる
10  0007 C604                ADD    A,4
11  0009 C605                ADD    A,5
12  000B C606                ADD    A,6
13  000D C607                ADD    A,7
14  000F C608                ADD    A,8
15  0011 C609                ADD    A,9
16  0013 C60A                ADD    A,10  <--- ここで1～10までの和がAレジスタに入っている
17  0015 76                  HALT        <--- CPUを止める

18                    ;****************************************
19                    ;SAMPLE2
20                    ;
21                    ;
22  0016 AF                  XOR    A
23  0017 060A                LD     B,10       カウンタの初期設定
24  0019 2601                LD     H,1        ポインタの初期設定
25  001B 84          LOOP1   ADD    A,H        A←A+H
26  001C 24                  INC    H          H←H+1
27  001D 10FC                DJNZ   LOOP1 <--- B←B−1 {Bがノンゼロなら LOOP1番地へ，ゼロなら次の番地へ
28  001F 76                  HALT
```

プログラム 2 （判断）

Aレジスタに入っているデータのビットパターンのうち“1”の数がいくつあるか数えます．結果が偶数ならGUSUR番地へ，奇数ならKISUR番地へジャンプするようにします．<SAMPLE 3>

もっとうまい方法があります．AとAのANDをとればAは変わらずに，そのときのAの1の数に応じてパリティフラグがセットされます．

```
AND   A            ←AとAのアンドによりFレジスタをセットします
JP    PE, GUSUR    ←PE，はパリティイーブンすなわちAレジスタの「1の
JP    KISUR          数が偶数であればジャンプせよ」の条件付ジャンプ命令です
```

これでも上と同じ働きをします．Fレジスタだけ変わりますが，他のレジスタの内容はこのルーチンに入る前のまま変化しません．

```
29                    ;************************************
30                    ;SAMPLE3
31                    ;
32                    ;
33          0200      GUSUR   EQU   0200H
34          025F      KISUR   EQU   025FH
35 0020 0600                  LD    B,0
36 0022 CB3F      LOOP2       SRL   A
37 0024 3001                  JR    NC,JP1
38 0026 04                    INC   B
39 0027 20F9      JP1         JR    NZ,LOOP2
40 0029 CB40                  BIT   0,B
41 002B CA0002                JP    Z,GUSUR
42 002E C35F02                JP    KISUR
```

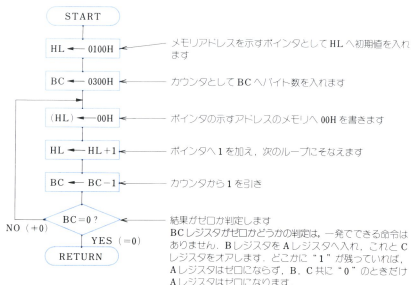
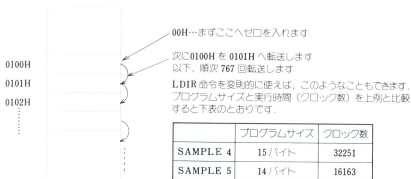
← EQU は GUSUR というラベル名を 0200 H 番地へ結びつけるためのアセンブラ命令

プログラム 3 （メモリクリアサブルーチン）

0100H 番地から 03FFH 番地までの 300H バイト(＝768 バイト)の RAM をゼロクリア（すべて 0 で埋めつくす）するサブルーチンを作ります．〈SAMPLE 4〉

別の方法もあります．〈SAMPLE 5〉

LDIR 命令を変則的に使えば，このようなこともできます．プログラムサイズと実行時間（クロック数）を上例と比較すると下表のとおりです．

	プログラムサイズ	クロック数
SAMPLE 4	15 バイト	32251
SAMPLE 5	14 バイト	16163

（プログラムサイズが小さく，実行が早いのがよいプログラムの条件です．ただし,あまり名人芸的なプログラムを作ると,あとで見たときに何をやっているのかわからなくなることがあります．メンテナンスやデバックをしやすいのも，よいプログラムの条件です）

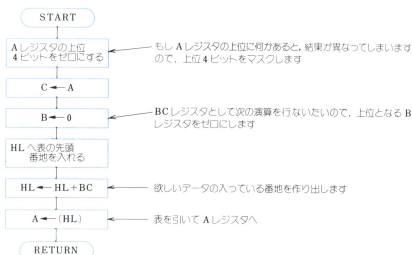
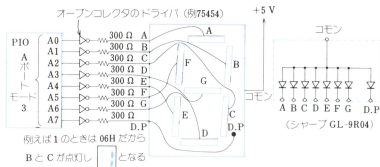
```
43                   ;***********************************
44                   ;SAMPLE4
45                   ;
46                   ;
47 0031 210001  MCLEAR LD    HL,0100H
48 0034 010003         LD    BC,0300H
49 0037 3600    ZEROM  LD    (HL),0
50 0039 23             INC   HL
51 003A 0B             DEC   BC
52 003B 78             LD    A,B      } B, C 共にゼロであることを調べる
53 003C B1             OR    C
54 003D 20F8           JR    NZ,ZEROM
55 003F C9             RET


56                   ;***********************************
57                   ;SAMPLE5
58                   ;
59                   ;
60 0040 210001  MCLR1  LD    HL,0100H
61 0043 110101         LD    DE,0101H
62 0046 01FF02         LD    BC,767 <---{ 10 進数で定義してよい
63 0049 3600           LD    (HL),0       マシン語は 02FF になっている
64 004B EDB0           LDIR
65 004D C9             RET
```

プログラム 4 （変換-テーブルサーチ）

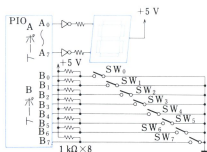
A レジスタの内容を表に従って変換します（下位4ビット対象）. <SAMPLE 6>

Aレジスタ	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
変　換　後	5C	06	5B	4F	66	6D	7D	27	7F	6F	77	7C	39	5E	79	71

変換前のデータが規則的で後は不規則です．したがって，前をメモリアドレスに対応させ，後をその内容とすれば，一発で表引きができます．もし相方共不規則なら一つ一つ一致するかどうか調べて行くサーチの手法をとらなければなりません．

これは4ビットのデータを数字表示LEDに,16進数表示するときに使うサブルーチンです．多桁表示したいときは，コモンを順次切り換えてダイナミック点灯させます．

```
66                  ;*********** **************************
67                  ;SAMPLE6
68                  ;
69                  ;
70 004E E60F    CONV    AND   0FH   ←――上位4ビットをマスク
71 0050 4F              LD    C,A
72 0051 0600            LD    B,0
73 0053 215900          LD    HL,TABLE
74 0056 09              ADD   HL,BC ←――HLに答の入っているアドレスを作成
75 0057 7E              LD    A,(HL)
76 0058 C9              RET
77 0059 5C      TABLE   DEFB  5CH
78 005A 06              DEFB  06H
79 005B 5B              DEFB  5BH
80 005C 4F              DEFB  4FH
81 005D 66              DEFB  66H
82 005E 6D              DEFB  6DH
83 005F 7D              DEFB  7DH
84 0060 27              DEFB  27H
85 0061 7F              DEFB  7FH
86 0062 6F              DEFB  6FH
87 0063 77              DEFB  77H
88 0064 7C              DEFB  7CH
89 0065 39              DEFB  39H
90 0066 5E              DEFB  5EH
91 0067 79              DEFB  79H
92 0068 71              DEFB  71H
```

(Lines 77–92:) テーブル（表）
（DEFBはオペランドに書かれた1バイトの値をそのままマシン語としてメモリへ定義するためのアセンブラ命令）

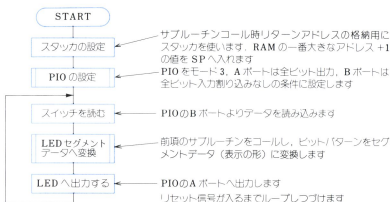
プログラム 5 （スイッチの表示）

PIO につながれた 8 個のスイッチのパターンを読み，前項のサブルーチンを使って LED に 16 進数表示させます．〈SAMPLE 7〉

PIO のポートアドレスは下表とします

A ポート　データ	D0
A ポート　コントロール	D1
B ポート　データ	D2
B ポート　コントロール	D3

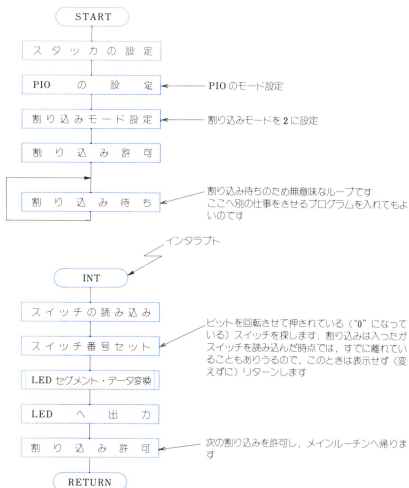
＝“1”

＝“0”

※SW_4～SW_7 は，この例では使わない（無視される）

IN 命令を実行するたびに，その時点の SW_0～SW_3 の状態が 16 進表現で LED に表示されます．

```
 93                     ;************* ***************************
 94                     ;SAMPLE7
 95                     ;
 96                     ;
 97          00D0       PIOAD1 EQU    0D0H          ポートアドレスのラベル名定義
 98          00D1       PIOAC1 EQU    0D1H          EQU は定義のためのアセンブラ命令
 99          00D2       PIOBD1 EQU    0D2H          (マシン語には変換されない)
100          00D3       PIOBC1 EQU    0D3H
101   0069   310000            LD     SP,0000H ←── スタッカ設定. FFFF H 番地より前へ
102   006C   218800            LD     HL,INTPA
103   006F   0603              LD     B,3           PIO A ポート. 制御語の書き込み
104   0071   0ED1              LD     C,PIOAC1
105   0073   EDB3              OTIR
106   0075   218B00            LD     HL,INTPB
107   0078   0603              LD     B,3           PIO B ポート. 制御語の書き込み
108   007A   0ED3              LD     C,PIOBC1
109   007C   EDB3              OTIR
110   007E   DBD2       LOOP7  IN     A,(PIOBD1)←── スイッチ読み込み
111   0080   CD4E00            CALL   CONV ←──────── データ変換をコール
112   0083   D3D0              OUT    (PIOAD1),A←── 出力
113   0085   C37E00            JP     LOOP7
114   0088   CF         INTPA  DEFB   0CFH ←──────── モード3
115   0089   00                DEFB   00H ←───────── 全ビット出力
116   008A   07                DEFB   07H ←───────── 割り込みなし
117   008B   CF         INTPB  DEFB   0CFH←───────── モード3
118   008C   FF                DEFB   0FFH←───────── 全ビット入力
119   008D   07                DEFB   07H ←───────── 割り込みなし
```

プログラム 6 （スイッチの表示-割り込み）

前項のスイッチを押しボタンスイッチとして，押されたスイッチ番号（SW_0 なら 0，SW_3 なら 3，SW_7 なら 7）を表示するように，割り込みを使ってプログラムを作ります．ただし，同時に押したときは，先に押したほうを優先します．全く同時なら番号の小さいほうを優先します．離しても次に押されるまで表示し続けます．〈SAMPLE 8〉

```
120                            ;**************************************
121                            ;SAMPLE8
122                            ;
123                            ;
124         00D0        PIOAD  EQU   0D0H          ┐
125         00D1        PIOAC  EQU   0D1H          │ ポートアドレスのラベル名定義
126         00D2        PIOBD  EQU   0D2H          │
127         00D3        PIOBC  EQU   0D3H          ┘
128  008E   310000             LD    SP,0000H        スタック設定．FFFF H 番地より前へ
129  0091   21AC00             LD    HL,INTLPA     ┐
130  0094   0603               LD    B,3           │ PIO A ポート．制御語の書き込み
131  0096   0ED1               LD    C,PIOAC       │
132  0098   EDB3               OTIR                ┘
133  009A   21AF00             LD    HL,INTLPB     ┐
134  009D   0605               LD    B,5           │ PIO B ポート．制御語の書き込み
135  009F   0ED3               LD    C,PIOBC       │
136  00A1   EDB3               OTIR                ┘
137  00A3   3E01               LD    A,01H         ┐ I レジスタへ 01 H
138  00A5   ED47               LD    I,A           ┘
139  00A7   ED5E               IM    2  <────────── 割り込みモード 2
140  00A9   FB                 EI  <─────────────── 割り込み許可
141  00AA   18FE        WAIT   JR    WAIT<───────── 割り込み待ちのループ
142  00AC   CF          INTLPA DEFB  0CFH<───────── モード 3
143  00AD   00                 DEFB  00H <───────── 全ビット出力
144  00AE   07                 DEFB  07H <───────── 割り込みなし
145  00AF   00          INTLPB DEFB  00H <───────── ベクトル 00 H
146  00B0   CF                 DEFB  0CFH<───────── モード 3
147  00B1   FF                 DEFB  0FFH<───────── 全ビット入力
148  00B2   97                 DEFB  97H <───────── ＊1 ┐
149  00B3   00                 DEFB  00H <───────── ＊2 │［注］下記参照
150                            ORG   0100H <─────── ＊3 │
151  0100   0201               DEFW  INT <───────── ＊4 ┘
152  0102   DBD2        INT    IN    A,(PIOBD)<──── スイッチの読み込み
153  0104   0E00               LD    C,0
154  0106   0608               LD    B,8
155  0108   1F          LOOP8  RRA <─────────────── 右にループし右端ビットをキャリへ
156  0109   3006               JR    NC,DISP<────── キャリを調べ"0"のとき表示へ
157  010B   0C                 INC   C
158  010C   10FA               DJNZ  LOOP8
159  010E   FB                 EI                  ┐ 8 ビット共調べて，すべて"1"のときここで
160  010F   ED4D               RETI                ┘ リターン
161  0111   79          DISP   LD    A,C
162  0112   CD4E00             CALL  CONV <──────── データ変換をコール
163  0115   D3D0               OUT   (PIOAD),A <─── LED へ出力
164  0117   FB                 EI
165  0118   ED4D               RETI
166                            ;**************************************
167                            END <─────────────── ソースプログラムの終りをアセンブラに知らせる
                                                    アセンブラ命令
```

＊1　割り込みあり，マスク制御語あり，オア，ローアクティブ
＊2　マスク制御語．全ビット割り込み発生
＊3　I レジスタとベクトルにより示される番地
＊4　割り込み処理ルーチンのエントリーアドレス 0102 番地
　　DEFW はオペランドの 2 バイトデータを定義するアセンブラ命令

付　録

付1　Z-80　命　令　表

○　n は　　8 ビット定数

○　lm は　16 ビット定数，l が上位 8 ビット，m が下位 8 ビット
（ニーモニックではラベル名を書いてもよい）

○　d は　　−128〜＋127 までのディスプレイスメント

○　e は　　−128〜＋127 までのディスプレイスメント．次の命令の先頭番地を 0 とする．
（ニーモニックではラベル名か絶対番地を書く．ただしアセンブラにより異なる場合もある）．

○　Cy は　　キャリ（C フラグ）

○　添字 $_H$ は上位 8 ビット，添字 $_L$ は下位 8 ビットを意味する．

○　ビット操作命令　SET　RES　BIT　で，ビット番号は下記のとおりである．

○　フラグ変化

×　不定
1　1 になる
0　0 になる
・　状態にしたがってセット，リセットされる
−　変化せず

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
ADC　A, n	CE n								7	8ビット加算キャリ付
ADC　A, A	8F									（アド・ウィズ・キャリ）
ADC　A, B	88									A←A＋ソース＋Cy
ADC　A, C	89									
ADC　A, D	8A								4	
ADC　A, E	8B									
ADC　A, H	8C	·	·	·	—	·	0	·		
ADC　A, L	8D									
ADC　A, (HL)	8E								7	
ADC　A, (IX+d)	DD 8E d								19	
ADC　A, (IY+d)	FD 8E d								19	
ADC　HL, BC	ED 4A									16ビット加算キャリ付
ADC　HL, DE	ED 5A	·	·	×	—	·	0	·	15	（アド・ウィズ・キャリ）
ADC　HL, HL	ED 6A									HL←HL＋ソース＋Cy
ADC　HL, SP	ED 7A									
ADD　A, n	C6 n								7	8ビット加算（アド）
ADD　A, A	87									
ADD　A, B	80									
ADD　A, C	81									A←A＋ソース
ADD　A, D	82								4	
ADD　A, E	83	·	·	·	—	·	0	·		
ADD　A, H	84									
ADD　A, L	85									
ADD　A, (HL)	86								7	
ADD　A, (IX+d)	DD 86 d								19	
ADD　A, (IY+d)	FD 86 d								19	
ADD　HL, BC	09									16ビット加算（アド）
ADD　HL, DE	19									
ADD　HL, HL	29								11	HL←HL＋ソース
ADD　HL, SP	39									
ADD　IX, BC	DD 09									
ADD　IX, DE	DD 19									
ADD　IX, IX	DD 29	—	—	×	—	—	0	·		IX←IX＋ソース
ADD　IX, SP	DD 39									
ADD　IY, BC	FD 09								15	
ADD　IY, DE	FD 19									
ADD　IY, IY	FD 29									IY←IY＋ソース
ADD　IY, SP	FD 39									
AND　n	E6 n								7	論理積（アンド）
AND　A	A7									
AND　B	A0									
AND　C	A1									A←A∧ソース
AND　D	A2								4	
AND　E	A3	·	·	1	·	—	0	0		
AND　H	A4									
AND　L	A5									
AND　(HL)	A6								7	
AND　(IX+d)	DD A6 d								19	
AND　(IY+d)	FD A6 d								19	

a	b	答
0	0	0
0	1	0
1	0	0
1	1	1

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
BIT 0,A	CB 47									ビットテスト
BIT 0,B	CB 40									
BIT 0,C	CB 41									
BIT 0,D	CB 42								8	ソースの第0ビットを調べ
BIT 0,E	CB 43									Zフラグを設定
BIT 0,H	CB 44	×	·	1	×	—	0	—		
BIT 0,L	CB 45									
BIT 0,(HL)	CB 46								12	
BIT 0,(IX+d)	DD CB d 46								20	
BIT 0,(IY+d)	FD CB d 46								20	
BIT 1,A	CB 4F									ビットテスト
BIT 1,B	CB 48									
BIT 1,C	CB 49									ソースの第1ビットを調べ
BIT 1,D	CB 4A								8	Zフラグを設定
BIT 1,E	CB 4B	×	·	1	×	—	0	—		
BIT 1,H	CB 4C									
BIT 1,L	CB 4D									
BIT 1,(HL)	CB 4E								12	
BIT 1,(IX+d)	DD CB d 4E								20	
BIT 1,(IY+d)	FD CB d 4E								20	
BIT 2,A	CB 57									ビットテスト
BIT 2,B	CB 50									
BIT 2,C	CB 51									ソースの第2ビットを調べ
BIT 2,D	CB 52								8	Zフラグを設定
BIT 2,E	CB 53	×	·	1	×	—	0	—		
BIT 2,H	CB 54									
BIT 2,L	CB 55									
BIT 2,(HL)	CB 56								12	
BIT 2,(IX+d)	DD CB d 56								20	
BIT 2,(IY+d)	FD CB d 56								20	
BIT 3,A	CB 5F									ビットテスト
BIT 3,B	CB 58									
BIT 3,C	CB 59									ソースの第3ビットを調べ
BIT 3,D	CB 5A								8	Zフラグを設定
BIT 3,E	CB 5B	×	·	1	×	—	0	—		
BIT 3,H	CB 5C									
BIT 3,L	CB 5D									
BIT 3,(HL)	CB 5E								12	
BIT 3,(IX+d)	DD CB d 5E								20	
BIT 3,(IY+d)	FD CB d 5E								20	
BIT 4,A	CB 67									ビットテスト
BIT 4,B	CB 60									
BIT 4,C	CB 61									ソースの第4ビットを調べ
BIT 4,D	CB 62								8	Zフラグを設定
BIT 4,E	CB 63	×	·	1	×	—	0	—		
BIT 4,H	CB 64									
BIT 4,L	CB 65									
BIT 4,(HL)	CB 66								12	
BIT 4,(IX+d)	DD CB d 66								20	
BIT 4,(IY+d)	FD CB d 66								20	

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
		フラグ変化			P/V					
BIT 5, A	CB 6F	×	·	1	×	−	0	−	8	ビットテスト
BIT 5, B	CB 68								8	ソースの第5ビットを調べ
BIT 5, C	CB 69								8	Zフラグを設定
BIT 5, D	CB 6A								8	
BIT 5, E	CB 6B								8	
BIT 5, H	CB 6C								8	
BIT 5, L	CB 6D								8	
BIT 5, (HL)	CB 6E								12	
BIT 5, (IX+d)	DD CB d 6E								20	
BIT 5, (IY+d)	FD CB d 6E								20	
BIT 6, A	CB 77	×	·	1	×	−	0	−	8	ビットテスト
BIT 6, B	CB 70								8	ソースの第6ビットを調べ
BIT 6, C	CB 71								8	Zフラグを設定
BIT 6, D	CB 72								8	
BIT 6, E	CB 73								8	
BIT 6, H	CB 74								8	
BIT 6, L	CB 75								8	
BIT 6, (HL)	CB 76								12	
BIT 6, (IX+d)	DD CB d 76								20	
BIT 6, (IY+d)	FD CB d 76								20	
BIT 7, A	CB 7F	×	·	1	×	−	0	−	8	ビットテスト
BIT 7, B	CB 78								8	ソースの第7ビットを調べ
BIT 7, C	CB 79								8	Zフラグを設定
BIT 7, D	CB 7A								8	
BIT 7, E	CB 7B								8	
BIT 7, H	CB 7C								8	
BIT 7, L	CB 7D								8	
BIT 7, (HL)	CB 7E								12	
BIT 7, (IX+d)	DD CB d 7E								20	
BIT 7, (IY+d)	FD CB d 7E								20	
CALL NZ, lm	C4 m l	−	−	−	−	−	−	−	成立時 17	サブルーチン・コール(条件付)
CALL Z, lm	CC m l									・条件が成立すれば戻り番地〔PC〕をスタッカへ PUSH し lm へジャンプ〔PC←lm〕
CALL NC, lm	D4 m l									
CALL C, lm	DC m l								不成立 10	・成立しなければ本命令は無視する
CALL PO, lm	E4 m l									
CALL PE, lm	EC m l									
CALL P, lm	F4 m l									
CALL M, lm	FC m l									
CALL lm	CD m l	−	−	−	−	−	−	−	17	サブルーチン・コール(無条件) PC をスタッカへ PUSH し PC←lm
CCF	3F	−	−	×	−	−	0	·	4	Cy を反転〔Cy←$\overline{Cy}$〕
CP n	FE n	·	·	·	−	·	1	·	7	比較（コンペア）
CP A	BF								4	A－ソースの演算をする
CP B	B8								4	Aの内容は変わらずフラグだけが変化する
CP C	B9								4	
CP D	BA								4	
CP E	BB								4	
CP H	BC								4	
CP L	BD								4	
CP (HL)	BE								7	
CP (IX+d)	DD BE d								19	
CP (IY+d)	FD BE d								19	

ニーモニック	マシン語	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
CPD	ED A9	×	·	×	−	·	1	−	16	比較（コンペア・ディクリメント） A−(HL) のフラグ変化のみ HL←HL−1　BC←BC−1
CPDR	ED B9	×	·	×	−	·	1	−	1バイトにつき 21 最終のみ 16	比較（コンペア・ディクリメント・リピート） CPD を A=(HL)〔Zフラグ=1〕 または　BC=0　〔Vフラグ=0〕 までくり返えす
CPI	ED A1	×	·	×	−	·	1	−	16	比較（コンペア・インクリメント） A−(HL) のフラグ変化のみ HL←HL+1　BC←BC−1
CPIR	ED B1	×	·	×	−	·	1	−	1バイトにつき 21 最終のみ 16	比較（コンペア・インクリメント・リピート） CPI を A=(HL)〔Zフラグ=1〕 または　BC=0　〔Vフラグ=0〕 までくり返えす
CPL	2F	−	−	1	−	−	1	−	4	コンプリメント Aレジスタに対しビット反転　0→1　1→0
DAA	27	·	·	·	·	−	−	·	4	デジマル・アジャスト・アキュムレータ Aレジスタに対し10進補正
DEC A DEC B DEC C DEC D DEC E DEC H DEC L DEC (HL) DEC (IX+d) DEC (IY+d)	3D 05 0D 15 1D 25 2D 35 DD 35 d FD 35 d	·	·	·	−	·	1	−	4（A〜L） 11 23 23	8ビットデクリメント ソース←ソース−1
DEC BC DEC DE DEC HL DEC SP DEC IX DEC IY	0B 1B 2B 3B DD 2B FD 2B	−	−	−	−	−	−	−	6（BC〜SP） 10 10	16ビットデクリメント ソース←ソース−1
DI	F3	−	−	−	−	−	−	−	4	割り込み禁止（ディセーブル・インタラプト）
DJNZ e	10 e	−	−	−	−	−	−	−	B=0 8 B≠0 13	（デクリメント・ジャンプノンゼロ） B←B−1　B≠0ならeバイトだけジャンプ〔PC←PC+e〕 B=0ならジャンプせず〔e=0と同じ〕
EI	FB	−	−	−	−	−	−	−	4	割り込み許可（イネーブル・インタラプト）
EX (SP), HL EX (SP), IX EX (SP), IY EX AF, AF' EX DE, HL	E3 DD E3 FD E3 08 EB	−	−	−	−	−	−	−	19 23 23 4 4	交換（エクスチェンジ） ソースとディスティネーションの内容を交換する
EXX	D9	−	−	−	−	−	−	−	4	BC　DE　HLの内容と BC' DE' HL' の内容を交換する

ニーモニック	マシン語	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
HALT	76	—	—	—	—	—	—	—	4	命令実行の進行を止めリセットまたは割り込み待ちとなる（ホルト）
IM 0	ED 46	—	—	—	—	—	—	—	8	割り込みモードを0に設定する
IM 1	ED 56								8	割り込みモードを1に設定する
IM 2	ED 5E								8	割り込みモードを2に設定する
INC A	3C	·	·	·	—	·	0	—	4	8ビットインクリメント ソース←ソース+1
INC B	04									
INC C	0C									
INC D	14									
INC E	1C									
INC H	24									
INC L	2C									
INC (HL)	34								11	
INC (IX+d)	DD 34 d								23	
INC (IY+d)	FD 34 d								23	
INC BC	03	—	—	—	—	—	—	—	6	16ビットインクリメント ソース←ソース+1
INC DE	13									
INC HL	23									
INC SP	33									
INC IX	DD 23								10	
INC IY	FD 23								10	
IN A,(C)	ED 78	·	·	0	·	—	0	—	12	入力 Cレジスタの内容番地のポートからディスティネーションのレジスタへ入力 [ディスティネーション←(C)]
IN B,(C)	ED 40									
IN C,(C)	ED 48									
IN D,(C)	ED 50									
IN E,(C)	ED 58									
IN H,(C)	ED 60									
IN L,(C)	ED 68									
IN A,(n)	DB n	—	—	—	—	—	—	—	11	入力　n番地のポートからAレジスタへ
IND	ED AA	×	·	×	×	—	×	×	16	イン・ディクリメント (HL)←(C)　HL←HL−1 B←B−1
INDR	ED BA	×	1	×	×	—	×	×	1バイトにつき 21 最終のみ 16	イン・ディクリメント・リピート INDをB=0までくり返えす
INI	ED A2	×	·	×	×	—	×	×	16	イン・インクリメント (HL)←(C)　HL←HL+1 B←B−1
INIR	ED B2	×	1	×	×	—	×	×	1バイトにつき 21 最終のみ 16	イン・インクリメント・リピート INIをB=0までくり返えす
JP (HL)	E9	—	—	—	—	—	—	—	4	ジャンプ（無条件） 各レジスタの内容番地へジャンプ [PC←HL]
JP (IX)	DD E9								8	
JP (IY)	FD E9								8	
JP lm	C3 m l								10	lm番地へジャンプ [PC←lm]

ニーモニック	マシン語	S	Z	H	P/V	N	C	所要クロックサイクル	動作
JP NZ, lm	C2 m l	—	—	—	—	—	—	10	ジャンプ（条件付）
JP Z, lm	CA m l								・条件が成立すれば lm 番地へジャンプ
JP NC, lm	D2 m l								（PC←lm）
JP C, lm	DA m l								・不成立なら本命令は無視する
JP PO, lm	E2 m l								
JP PE, lm	EA m l								
JP P, lm	F2 m l								
JP M, lm	FA m l								
JR e	18 e	—	—	—	—	—	—	12	ジャンプ・リラティブ（無条件） e バイト先へジャンプする [PC←PC＋e]
JR NZ, e	20 e	—	—	—	—	—	—	条件成立 12 条件不成立 7	ジャンプ・リラティブ(条件付)
JR Z, e	28 e								・条件が成立すればe バイト先へジャンプ
JR NC, e	30 e								[PC←PC＋e]
JR C, e	38 e								・不成立なら本命令は無視する
LD A, n	3E n	—	—	—	—	—	—	7	8ビット転送（ロード）
LD A, A	7F							4	
LD A, B	78							4	A←ソース
LD A, C	79							4	
LD A, D	7A							4	
LD A, E	7B							4	
LD A, H	7C							4	
LD A, L	7D							4	
LD A, (lm)	3A m l							13	
LD A, (BC)	0A							7	
LD A, (DE)	1A							7	
LD A, (HL)	7E							7	
LD A, (IX+d)	DD 7E d							19	
LD A, (IY+d)	FD 7E d							19	
LD A, I	ED 57	・	・	0	IFF	0	—	9	8ビット転送　A←I
LD A, R	ED 5F							9	A←R
									IFF：0のとき割り込み禁止（DI） 1のとき割り込み可（EI）になっている LD A, I　LD A, R ではこの値が P/V にコピーされる
LD B, n	06 n	—	—	—	—	—	—	7	8ビット転送（ロード）
LD B, A	47							4	
LD B, B	40							4	B←ソース
LD B, C	41							4	
LD B, D	42							4	
LD B, E	43							4	
LD B, H	44							4	
LD B, L	45							4	
LD B, (HL)	46							7	
LD B, (IX+d)	DD 46 d							19	
LD B, (IY+d)	FD 46 d							19	

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V	N	C	所要クロックサイクル	動作
LD C, n	0E n							7	8ビット転送（ロード）
LD C, A	4F								
LD C, B	48								
LD C, C	49								C←ソース
LD C, D	4A							4	
LD C, E	4B	—	—	—	—	—	—		
LD C, H	4C								
LD C, L	4D								
LD C, (HL)	4E							7	
LD C, (IX+d)	DD 4E d							19	
LD C, (IY+d)	FD 4E d							19	
LD D, n	16 n							7	8ビット転送（ロード）
LD D, A	57								
LD D, B	50								
LD D, C	51								D←ソース
LD D, D	52							4	
LD D, E	53	—	—	—	—	—	—		
LD D, H	54								
LD D, L	55								
LD D, (HL)	56							7	
LD D, (IX+d)	DD 56 d							19	
LD D, (IY+d)	FD 56 d							19	
LD E, n	1E n							7	8ビット転送（ロード）
LD E, A	5F								
LD E, B	58								
LD E, C	59								E←ソース
LD E, D	5A							4	
LD E, E	5B	—	—	—	—	—	—		
LD E, H	5C								
LD E, L	5D								
LD E, (HL)	5E							7	
LD E, (IX+d)	DD 5E d							19	
LD E, (IY+d)	FD 5E d							19	
LD H, n	26 n							7	8ビット転送（ロード）
LD H, A	67								
LD H, B	60								
LD H, C	61								H←ソース
LD H, D	62							4	
LD H, E	63	—	—	—	—	—	—		
LD H, H	64								
LD H, L	65								
LD H, (HL)	66							7	
LD H, (IX+d)	DD 66 d							19	
LD H, (IY+d)	FD 66 d							19	

ニーモニック	マシン語	S	Z	H	P/V	N	C	所要クロックサイクル	動作
LD L, n	2E n							7	8ビット転送（ロード）
LD L, A	6F								
LD L, B	68								L←ソース
LD L, C	69								
LD L, D	6A							4	
LD L, E	6B	—	—	—	—	—	—		
LD L, H	6C								
LD L, L	6D								
LD L, (HL)	6E							7	
LD L, (IX+d)	DD 6E d							19	
LD L, (IY+d)	FD 6E d							19	
LD I, A	ED 47	—	—	—	—	—	—	9	8ビット転送　I←A
LD R, A	ED 4F							9	R←A
LD (lm), A	32 m l							13	8ビット転送 (lm)←A
LD (BC), A	02	—	—	—	—	—	—	7	(BC)←A
LD (DE), A	12							7	(DE)←A
LD (HL), n	36 n							10	8ビット転送（ロード）
LD (HL), A	77								
LD (HL), B	70								
LD (HL), C	71	—	—	—	—	—	—		(HL)←ソース
LD (HL), D	72							7	
LD (HL), E	73								
LD (HL), H	74								
LD (HL), L	75								
LD (IX+d), n	DD 36 d n								8ビット転送（ロード）
LD (IX+d), A	DD 77 d								
LD (IX+d), B	DD 70 d								
LD (IX+d), C	DD 71 d								(IX+d)←ソース
LD (IX+d), D	DD 72 d	—	—	—	—	—	—		
LD (IX+d), E	DD 73 d							19	
LD (IX+d), H	DD 74 d								
LD (IX+d), L	DD 75 d								
LD (IY+d), n	FD 36 d n								8ビット転送（ロード）
LD (IY+d), A	FD 77 d								
LD (IY+d), B	FD 70 d								
LD (IY+d), C	FD 71 d								(IY+d)←ソース
LD (IY+d), D	FD 72 d	—	—	—	—	—	—		
LD (IY+d), E	FD 73 d							19	
LD (IY+d), H	FD 74 d								
LD (IY+d), L	FD 75 d								
LD BC, lm	01 m l	—	—	—	—	—	—	10	16ビット転送　〔lm：定数 (lm)：メモリの内容〕
LD BC, (lm)	ED 4B m l							20	BC←ソース　メモリからレジスタの場合
LD DE, lm	11 m l	—	—	—	—	—	—	10	16ビット転送　たとえば
LD DE, (lm)	ED 5B m l							20	DE←ソース　HL, (lm)では L←(lm)
LD HL, lm	21 m l	—	—	—	—	—	—	10	16ビット転送　H←(lm+1)
LD HL, (lm)	2A m l							16	HL←ソース　となる

ニーモニック	マシン語	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
LD SP, lm	31 m l	−	−	−	−	−	−	−	10	16ビット転送
LD SP, (lm)	ED 7B m l								20	
LD SP, HL	F9								6	SP←ソース
LD SP, IX	DD F9								10	
LD SP, IY	FD F9								10	
LD IX, lm	DD 21 m l	−	−	−	−	−	−	−	14	16ビット転送
LD IX, (lm)	DD 2A m l								20	IX←ソース
LD IY, lm	FD 21 m l	−	−	−	−	−	−	−	14	16ビット転送
LD IY, (lm)	FD 2A m l								20	IY←ソース
LD (lm), BC	ED 43 m l	−	−	−	−	−	−	−	20	16ビット転送（ロード）
LD (lm), DE	ED 53 m l								20	
LD (lm), HL	22 m l								16	(lm)←ソースL
LD (lm), SP	ED 73 m l								20	(lm+1)←ソースH
LD (lm), IX	DD 22 m l								20	
LD (lm), IY	FD 22 m l								20	
LDD	ED A8	×	×	0	−	·	0	−	16	ブロック転送（ロード・ディクリメント） (DE)←(HL)　DE←DE−1 HL←HL−1　BC←BC−1
LDDR	ED B8	×	×	0	−	0	0	−	1バイトにつき 21 最終のみ 16	ブロック転送 （ロード・ディクリメント・リピート） LDDをBC＝0までくり返えす
LDI	ED A0	×	×	0	−	·	0	−	16	ブロック転送（ロード・インクリメント） (DE)←(HL)　DE←DE＋1 HL←HL＋1　BC←BC−1
LDIR	ED B0	×	×	0	−	0	0	−	1バイトにつき 21 最終のみ 16	ブロック転送 （ロード・インクリメント・リピート） LDIをBC＝0までくり返えす
NEG	ED 44	·	·	·	−	·	1	·	8	ニゲイト　Zの補数をとる A←0−A
NOP	00	−	−	−	−	−	−	−	4	何もしないで次へ （ノーオペレーション）
OR n	F6 n	·	·	0	·	−	0	0	7	論理和（オア）
OR A	B7								4	
OR B	B0								4	A←A∨ソース
OR C	B1								4	
OR D	B2								4	
OR E	B3								4	
OR H	B4								4	
OR L	B5								4	
OR (HL)	B6								7	
OR (IX+d)	DD B6 d								19	
OR (IY+d)	FD B6 d								19	

a	b	答
0	0	0
0	1	1
1	0	1
1	1	1

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
OUT　(C), A	ED 79									出力
OUT　(C), B	ED 41									
OUT　(C), C	ED 49									各レジスタの内容をCレジスタの内容番地
OUT　(C), D	ED 51	−	−	−	−	−	−	−	12	のポートへ出力
OUT　(C), E	ED 59									[(C)←ソース]
OUT　(C), H	ED 61									
OUT　(C), L	ED 69									
OUT　(n), A	D3 n	−	−	−	−	−	−	−	11	出力　Aレジスタの内容をn番地のポートへ
OUTD	ED AB	×	·	×	×	−	×	×	16	アウト・ディクリメント (C)←(HL)　HL←HL−1 B←B−1
OTDR	ED BB	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	アウト・ディクリメント・リピート OUTDをB＝0までくり返えす
OUTI	ED A3	×	·	×	×	−	×	×	16	アウト・インクリメント (C)←(HL)　HL←HL＋1 B←B−1
OTIR	ED B3	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	アウト・インクリメント・リピート OUTIをB＝0までくり返えす
POP　AF	F1									16ビット転送（ポップ）
POP　BC	C1									スタッカからレジスタへ転送
POP　DE	D1	−	−	−	−	−	−	−	10	ディスティネーション$_L$←(SP)
POP　HL	E1									ティスティネーション$_H$←(SP＋1)
POP　IX	DD E1								14	SP←SP＋2
POP　IY	FD E1								14	
PUSH　AF	F5									16ビット転送（プッシュ）
PUSH　BC	C5									レジスタからスタッカへ転送
PUSH　DE	D5	−	−	−	−	−	−	−	11	(SP−1)←ソース$_H$
PUSH　HL	E5									(SP−2)←ソース$_L$
PUSH　IX	DD E5								15	SP←SP−2
PUSH　IY	FD E5								15	
RES　0, A	CB 87									ビットリセット
RES　0, B	CB 80									
RES　0, C	CB 81									ソースの第0ビット←0
RES　0, D	CB 82								8	
RES　0, E	CB 83	−	−	−	−	−	−	−		
RES　0, H	CB 84									
RES　0, L	CB 85									
RES　0, (HL)	CB 86								15	
RES　0, (IX+d)	DD CB d 86								23	
RES　0, (IY+d)	FD CB d 86								23	

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
		フラグ変化			P/V					
RES 1,A	CB 8F									ビットリセット
RES 1,B	CB 88									
RES 1,C	CB 89									ソースの第1ビット←0
RES 1,D	CB 8A								8	
RES 1,E	CB 8B	−	−	−	−	−	−	−		
RES 1,H	CB 8C									
RES 1,L	CB 8D									
RES 1,(HL)	CB 8E								15	
RES 1,(IX+d)	DD CB d 8E								23	
RES 1,(IY+d)	FD CB d 8E								23	
RES 2,A	CB 97									ビットリセット
RES 2,B	CB 90									
RES 2,C	CB 91									ソースの第2ビット←0
RES 2,D	CB 92								8	
RES 2,E	CB 93	−	−	−	−	−	−	−		
RES 2,H	CB 94									
RES 2,L	CB 95									
RES 2,(HL)	CB 96								15	
RES 2,(IX+d)	DD CB d 96								23	
RES 2,(IY+d)	FD CB d 96								23	
RES 3,A	CB 9F									ビットリセット
RES 3,B	CB 98									
RES 3,C	CB 99									ソースの第3ビット←0
RES 3,D	CB 9A								8	
RES 3,E	CB 9B	−	−	−	−	−	−	−		
RES 3,H	CB 9C									
RES 3,L	CB 9D									
RES 3,(HL)	CB 9E								15	
RES 3,(IX+d)	DD CB d 9E								23	
RES 3,(IY+d)	FD CB d 9E								23	
RES 4,A	CB A7									ビットリセット
RES 4,B	CB A0									
RES 4,C	CB A1									ソースの第4ビット←0
RES 4,D	CB A2								8	
RES 4,E	CB A3	−	−	−	−	−	−	−		
RES 4,H	CB A4									
RES 4,L	CB A5									
RES 4,(HL)	CB A6								15	
RES 4,(IX+d)	DD CB d A6								23	
RES 4,(IY+d)	FD CB d A6								23	
RES 5,A	CB AF									ビットリセット
RES 5,B	CB A8									
RES 5,C	CB A9									ソースの第5ビット←0
RES 5,D	CB AA								8	
RES 5,E	CB AB	−	−	−	−	−	−	−		
RES 5,H	CB AC									
RES 5,L	CB AD									
RES 5,(HL)	CB AE								15	
RES 5,(IX+d)	DD CB d AE								23	
RES 5,(IY+d)	FD CB d AE								23	

ニーモニック	マシン語	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
RES 6, A	CB B7	—	—	—	—	—	—	—	8	ビットリセット ソースの第6ビット←0
RES 6, B	CB B0									
RES 6, C	CB B1									
RES 6, D	CB B2									
RES 6, E	CB B3									
RES 6, H	CB B4									
RES 6, L	CB B5									
RES 6, (HL)	CB B6								15	
RES 6, (IX+d)	DD CB d B6								23	
RES 6, (IY+d)	FD CB d B6								23	
RES 7, A	CB BF	—	—	—	—	—	—	—	8	ビットリセット ソースの第7ビット←0
RES 7, B	CB B8									
RES 7, C	CB B9									
RES 7, D	CB BA									
RES 7, E	CB BB									
RES 7, H	CB BC									
RES 7, L	CB BD									
RES 7, (HL)	CB BE								15	
RES 7, (IX+d)	DD CB d BE								23	
RES 7, (IY+d)	FD CB d BE								23	
RET	C9	—	—	—	—	—	—	—	10	サブルーチンからのリターン PCへスタックよりPOP
RET NZ	C0	—	—	—	—	—	—	—	条件成立 11 不成立 5	条件付リターン ・条件が成立すれば POP PC ・不成立なら本命令は無視する
RET Z	C8									
RET NC	D0									
RET C	D8									
RET PO	E0									
RET PE	E8									
RET P	F0									
RET M	F8									
RETI	ED 4D	—	—	—	—	—	—	—	14	割り込み処理からのリターン POP PC
RETN	ED 45	—	—	—	—	—	—	—	14	ノンマスカブル割り込み処理からのリターン POP PC
RLA	17	—	—	0	—	—	0	・	4	ローテート・レフト・アキュムレータ RL A と同じ
RLCA	07	—	—	0	—	—	0	・	4	ローテート・レフト・サーキュラ・アキュムレータ RLC A と同じ
RRA	1F	—	—	0	—	—	0	・	4	ローテート・ライト・アキュムレータ RR A と同じ
RRCA	0F	—	—	0	—	—	0	・	4	ローテート・ライト・サーキュラ・アキュムレータ RRC A と同じ

ニーモニック	マシン語	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
RL　A	CB 17									ローテート・レフト
RL　B	CB 10									
RL　C	CB 11									
RL　D	CB 12								8	
RL　E	CB 13									Cy ← 7　　0 ←
RL　H	CB 14	・	・	0	・	−	0	・		
RL　L	CB 15									
RL　(HL)	CB 16								15	
RL　(IX+d)	DD CB d 16								23	
RL　(IY+d)	FF CB d 16								23	
RLC　A	CB 07									ローテート・レフト・サーキュラ
RLC　B	CB 00									
RLC　C	CB 01									
RLC　D	CB 02								8	
RLC　E	CB 03	・	・	0	・	−	0	・		Cy ← 7　　0 ←
RLC　H	CB 04									
RLC　L	CB 05									
RLC　(HL)	CB 06								15	
RLC　(IX+d)	DD CB d 06								23	
RLC　(IY+d)	FD CB d 06								23	
RR　A	CB 1F									ローテート・ライト
RR　B	CB 18									
RR　C	CB 19									
RR　D	CB 1A								8	
RR　E	CB 1B									
RR　H	CB 1C	・	・	0	・	−	0	・		→ Cy → 7　　0
RR　L	CB 1D									
RR　(HL)	CB 1E								15	
RR　(IX+d)	DD CB d 1E								23	
RR　(IY+d)	FD CB d 1E								23	
RRC　A	CB 0F									ローテート・ライト・サーキュラ
RRC　B	CB 08									
RRC　C	CB 09									
RRC　D	CB 0A								8	
RRC　E	CB 0B									
RRC　H	CB 0C	・	・	0	・	−	0	・		→ Cy → 7　　0
RRC　L	CB 0D									
RRC　(HL)	CB 0E								15	
RRC　(IX+d)	DD CB d 0E								23	
RRC　(IY+d)	FD CB d 0E								23	
RLD	ED 6F	・	・	0	・	−	0	−	18	ローテート・レフト・ディジット A　(HL) 7 4 3 0　7 4 3 0
RRD	ED 67	・	・	0	・	−	0	−	18	ローテート・ライト・ディジット A　(HL) 7 4 3 0　7 4 3 0

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V P	P/V V	N	C		
RST 00H	C7	—	—	—	—	—	—	—	11	リスタート　0000H～0038Hのいずれかの番地に対するCALL
RST 08H	CF									
RST 10H	D7									
RST 18H	DF									
RST 20H	E7									
RST 28H	EF									
RST 30H	F7									
RST 38H	FF									
SBC A, n	DE n	·	·	·	—	·	1	·	7	8ビット引き算（キャリ付）（サブトラクト・ウィズ・キャリ）　A←A－ソース－Cy
SBC A, A	9F								4	
SBC A, B	98									
SBC A, C	99									
SBC A, D	9A									
SBC A, E	9B									
SBC A, H	9C									
SBC A, L	9D									
SBC A, (HL)	9E								7	
SBC A, (IX+d)	DD 9E d								19	
SBC A, (IY+d)	FD 9E d								19	
SBC HL, BC	ED 42	·	·	×	—	·	1	·	15	16ビット引き算（キャリ付）（サブトラクト・ウィズ・キャリ）　HL←HL－ソース－Cy
SBC HL, DE	ED 52									
SBC HL, HL	ED 62									
SBC HL, SP	ED 72									
SCF	37	—	—	0	—	—	0	1	4	セット・キャリフラグ　Cy←1
SET 0, A	CB C7	—	—	—	—	—	—	—	8	ビットセット　ソースの第0ビット←1
SET 0, B	CB C0									
SET 0, C	CB C1									
SET 0, D	CB C2									
SET 0, E	CB C3									
SET 0, H	CB C4									
SET 0, L	CB C5									
SET 0, (HL)	CB C6								15	
SET 0, (IX+d)	DD CB d C6								23	
SET 0, (IY+d)	FD CB d C6								23	
SET 1, A	CB CF	—	—	—	—	—	—	—	8	ビットセット　ソースの第1ビット←1
SET 1, B	CB C8									
SET 1, C	CB C9									
SET 1, D	CB CA									
SET 1, E	CB CB									
SET 1, H	CB CC									
SET 1, L	CB CD									
SET 1, (HL)	CB CE								15	
SET 1, (IX+d)	DD CB d CE								23	
SET 1, (IY+d)	FD CB d CE								23	

ニーモニック	マシン語	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
SET 2, A	CB D7									ビットセット
SET 2, B	CB D0									
SET 2, C	CB D1									
SET 2, D	CB D2								8	ソースの第2ビット←1
SET 2, E	CB D3	—	—	—	—	—	—	—		
SET 2, H	CB D4									
SET 2, L	CB D5									
SET 2, (HL)	CB D6								15	
SET 2, (IX+d)	DD CB d D6								23	
SET 2, (IY+d)	FD CB d D6								23	
SET 3, A	CB DF									ビットセット
SET 3, B	CB D8									
SET 3, C	CB D9									
SET 3, D	CB DA								8	ソースの第3ビット←1
SET 3, E	CB DB	—	—	—	—	—	—	—		
SET 3, H	CB DC									
SET 3, L	CB DD									
SET 3, (HL)	CB DE								15	
SET 3, (IX+d)	DD CB d DE								23	
SET 3, (IY+d)	FD CB d DE								23	
SET 4, A	CB E7									ビットセット
SET 4, B	CB E0									
SET 4, C	CB E1									
SET 4, D	CB E2								8	ソースの第4ビット←1
SET 4, E	CB E3	—	—	—	—	—	—	—		
SET 4, H	CB E4									
SET 4, L	CB E5									
SET 4, (HL)	CB E6								15	
SET 4, (IX+d)	DD CB d E6								23	
SET 4, (IY+d)	FD CB d E6								23	
SET 5, A	CB EF									ビットセット
SET 5, B	CB E8									
SET 5, C	CB E9									
SET 5, D	CB EA								8	ソースの第5ビット←1
SET 5, E	CB EB	—	—	—	—	—	—	—		
SET 5, H	CB EC									
SET 5, L	CB ED									
SET 5, (HL)	CB EE								15	
SET 5, (IX+d)	DD CB d EE								23	
SET 5, (IY+d)	FD CB d EE								23	
SET 6, A	CB F7									ビットセット
SET 6, B	CB F0									
SET 6, C	CB F1									
SET 6, D	CB F2								8	ソースの第6ビット←1
SET 6, E	CB F3	—	—	—	—	—	—	—		
SET 6, H	CB F4									
SET 6, L	CB F5									
SET 6, (HL)	CB F6								15	
SET 6, (IX+d)	DD CB d F6								23	
SET 6, (IY+d)	FD CB d F6								23	

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
		フラグ変化			P/V					
SET 7, A	CB FF									ビットセット
SET 7, B	CB F8									
SET 7, C	CB F9									ソースの第7ビット←1
SET 7, D	CB FA								8	
SET 7, E	CB FB	—	—	—	—	—	—	—		
SET 7, H	CB FC									
SET 7, L	CB FD									
SET 7, (HL)	CB FE								15	
SET 7, (IX+d)	DD CB d FE								23	
SET 7, (IY+d)	FD CB d FE								23	
SLA A	CB 27									シフト・レフト・アリスメチック
SLA B	CB 20									
SLA C	CB 21									
SLA D	CB 22								8	
SLA E	CB 23	・	・	0	・	—	0	・		Cy ← 7　0 ← 0
SLA H	CB 24									
SLA L	CB 25									
SLA (HL)	CB 26								15	
SLA (IX+d)	DD CB d 26								23	
SLA (IY+d)	FD CB d 26								23	
SRA A	CB 2F									シフト・ライト・アリスメチック
SRA B	CB 28									
SRA C	CB 29									
SRA D	CB 2A								8	
SRA E	CB 2B	・	・	0	・	—	0	・		Cy　7　0
SRA H	CB 2C									
SRA L	CB 2D									
SRA (HL)	CB 2E								15	
SRA (IX+d)	DD CB d 2E								23	
SRA (IY+d)	FD CB d 2E								23	
SRL A	CB 3F									シフト・ライト・ロジカル
SRL B	CB 38									
SRL C	CB 39									
SRL D	CB 3A								8	
SRL E	CB 3B	・	・	0	・	—	0	・		Cy　7　0　0
SRL H	CB 3C									
SRL L	CB 3D									
SRL (HL)	CB 3E								15	
SRL (IX+d)	DD CB d 3E								23	
SRL (IY+d)	FD CB d 3E								23	
SUB n	D6 n								7	8ビット引き算
SUB A	97									（サブトラクト）
SUB B	90									
SUB C	91									A←A－ソース
SUB D	92								4	
SUB E	93	・	・	・	—	・	1	・		
SUB H	94									
SUB L	95									
SUB (HL)	96								7	
SUB (IX+d)	DD 96 d								19	
SUB (IY+d)	FD 96 d								19	

ニーモニック	マシン語	S	Z	H	P/V	N	C	所要クロックサイクル	動作
XOR n	EE n	·	·	0	·	—	0 0	7	排他的論理和(エクスクルーシブ・オア)
XOR A	AF							4	A←A⊕ソース
XOR B	A8							4	
XOR C	A9							4	
XOR D	AA							4	
XOR E	AB							4	
XOR H	AC							4	
XOR L	AD							4	
XOR (HL)	AE							7	
XOR (IX+d)	DD AE d							19	
XOR (IY+d)	FD AE d							19	

a	b	答
0	0	0
0	1	1
1	0	1
1	1	0

フラグ変化表

	命令		S	Z	H	P/V P	P/V V	N	C	コメント
1	8ビット加算	ADD ADC	·	·	·	—	·	0	·	
2	8ビット減算系	SUB SBC CP NEG	·	·	·	—	·	1	·	
3	論理積	AND	·	·	1	·	—	0	0	Cフラグを0にするとき
4	論理和	OR XOR	·	·	0	·	—	0	0	
5	8ビットインクリメント	INC	·	·	·	—	·	0	—	16ビットINC DECでは変化しない
6	8ビットデクリメント	DEC	·	·	·	—	·	1	—	
7	16ビット加算	ADD	—	—	×	—	—	0	·	
8	16ビットキャリ付加算	ADC	·	·	×	—	·	0	·	
9	16ビットキャリ付減算	SBC	·	·	×	—	·	1	·	
10	ローテート・アキュムレータ	RLA RLCA RRA RRCA	—	—	0	—	—	0	·	
11	ローテート	RL RLC RR RRC	·	·	0	·	—	0	·	
12	シフト	SLA SRA SRL	·	·	0	·	—	0	·	
13	ローテート・ディジット	RLD RRD	·	·	0	·	—	0	—	
14	10進補正	DAA	·	·	·	·	—	—	·	
15	ビット反転	CPL	—	—	1	—	—	1	—	
16	セットキャリ	SCF	—	—	0	—	—	0	1	
17	キャリ反転	CCF	—	—	×	—	—	0	·	
18	間接アドレシング入力	IN r,(C)	·	·	0	·	—	0	—	IN A,(n)では変化しない
19	ブロック入力出力	INI IND OUTI OUTD	×	·	×	×	—	×	×	
20	リピート入力出力	INIR INDR OTIR OTDR	×	1	×	×	—	×	×	
21	ブロック転送	LDI LDD	×	×	0	—	·	0	—	
22	リピート転送	LDIR LDDR	×	×	0	—	0	0	—	
23	ブロックサーチ	CPI CPD CPIR CPDR	×	·	×	—	·	1	—	
24	Iレジスタ,Rレジスタ	LD A,I LD A,R	·	·	0	IFF		0	—	P/VにIFFがコピーされる
25	ビットテスト	BIT	×	·	1	×	—	0	—	

× 不定
1 1になる
0 0になる
· 状態にしたがってセット，リセットされる
— 変化せず

IFF：0のとき割り込み禁止(DI)
1のとき割り込み可(EI)
になっている
LD A,I　LD A,R
ではこの値がP/Vにコピーされる

付2　Z-80 規格表（参考）

（1） CPU

絶対最大定格

項　目	記 号	定 格 値	単 位
入力電圧	V_{IN}	−0.3 ～ +7	V
出力電圧	V_{OUT}	−0.3 ～ +7	V
動作温度	T_{opr}	0 ～ +70	℃
保存温度	T_{stg}	−65 ～ +150	℃

電気的特性

DC 特性

（Ta ＝ 0 ℃～＋70 ℃，V_{CC} ＝＋5 V ± 5 %）

記 号	項　目	最 小 値	最 大 値	単位	測 定 条 件
V_{ILC}	クロック“L”入力電圧	−0.3	0.45	V	
V_{IHC}	クロック“H”入力電圧	V_{CC}−0.6	V_{CC}+0.3	V	
V_{IL}	“L”入力電圧	−0.3	0.8	V	
V_{IH}	“H”入力電圧	2.0	V_{CC}	V	
V_{OL}	“L”出力電圧		0.4	V	I_{OL} = 1.8mA
V_{OH}	“H”出力電圧	2.4		V	I_{OH} = −250μA
I_{CC}	消費電流		150 200	mA	（上段 Z-80 CPU 下段 Z-80A CPU）
I_{LI}	入力リーク電流		10	μA	V_{IN}=0～V_{CC}
I_{LOH}	トライステート出力リーク電流		10	μA	V_{OUT}=2.4V～V_{CC}
I_{LOL}	トライステート出力リーク電流		−10	μA	V_{OUT}= 0.4 V
I_{LD}	入力時のデータ・バスのリーク電流		±10	μA	$0 \leqq V_{IN} \leqq V_{CC}$

端子容量

（Ta ＝＋25 ℃，f ＝ 1 MHz）

記 号	項　目	最大値	単 位	測 定 条 件
C_{Φ}	クロック入力容量	50	pF	被測定端子以外のすべての端子は接地
C_{IN}	入力容量	8	pF	
C_{OUT}	出力容量	12	pF	

AC 特性

（Ta ＝0℃～＋70℃ ，V_{CC} ＝＋5V±5%）

信　号	記　号	パ　ラ　メ　ー　タ	Z-80 CPU 最小値	Z-80 CPU 最大値	Z-80A CPU 最小値	Z-80A CPU 最大値	単位	測定条件
Φ	t_C	クロック周期	0.4	200	0.25	200	μs	
	$t_{W(\Phi H)}$	クロック・パルス幅（“H”）	180		110		ns	
	$t_{W(\Phi L)}$	クロック・パルス幅（“L”）	180	2000	110	2000	ns	
	t_r ,t_f	クロックの立ち上がり・立ち下がり時間		30		30	ns	
A_0-A_{15}	t_D(AD)	クロックの立ち上がりから出力までの遅延		145		110	ns	C_L=50pF
	t_F(AD)	出力がフロート状態になるまでの遅延		110		90	ns	
	t_{acm}	$\overline{MREQ}$に先立つ出力確定時間（メモリ・サイクル）	〔1〕		〔1〕		ns	
	t_{aci}	$\overline{IORQ}$,$\overline{RD}$または$\overline{WR}$に先立つ出力確定時間（入出力サイクル）	〔2〕		〔2〕		ns	
	t_{ca}	$\overline{RD}$,$\overline{WR}$,$\overline{IORQ}$または$\overline{MREQ}$からの出力保持時間	〔3〕		〔3〕		ns	
	t_{caf}	$\overline{RD}$または$\overline{WR}$からの出力保持時間（フロート状態への遷移時）	〔4〕		〔4〕		ns	
D_0-D_7	t_D(D)	クロックの立ち下がりから出力までの遅延		230		150	ns	C_L=50pF
	t_F(D)	出力がフロート状態になるまでの遅延（書き込みサイクル）		90		90	ns	
	$t_{S\Phi}$(D)	クロックの立ち上がりに対するセットアップ時間（M1サイクル）	50		35		ns	
	$t_{S\overline{\Phi}}$(D)	クロックの立ち下がりに対するセットアップ時間（M2～M5サイクル）	60		50		ns	
	t_{dcm}	$\overline{WR}$に先立つ出力確定時間（メモリ・サイクル）	〔5〕		〔5〕		ns	
	t_{dci}	$\overline{WR}$に先立つ出力確定時間（入出力サイクル）	〔6〕		〔6〕		ns	
	t_{cdf}	$\overline{WR}$からの出力保存時間	〔7〕		〔7〕		ns	
	t_H	ホールド時間	0		0		ns	
$\overline{MREQ}$	$t_{DL\overline{\Phi}}$(MR)	クロックの立ち下がりから$\overline{MREQ}$＝“L”になるまでの遅延		100		85	ns	C_L=50pF
	$t_{DH\Phi}$(MR)	クロックの立ち上がりから$\overline{MREQ}$＝“H”になるまでの遅延（M1サイクル）		100		85	ns	

信号	記号	パラメータ	Z-80 CPU 最小値	Z-80 CPU 最大値	Z-80A CPU 最小値	Z-80A CPU 最大値	単位	測定条件
$\overline{MREQ}$	$t_{DH\overline{\Phi}}$ (MR)	クロックの立ち下がりから$\overline{MREQ}$＝"H"になるまでの遅延（M2～M5サイクル）		100		85	ns	C_L=50pF
	$t_W(\overline{MR_L})$	$\overline{MREQ}$のパルス幅("L")	〔8〕		〔8〕		ns	
	$t_W(\overline{MR_H})$	$\overline{MREQ}$のパルス幅("H")	〔9〕		〔9〕		ns	
$\overline{IORQ}$	$t_{DL\Phi}$ (IR)	クロックの立ち上がりから$\overline{IORQ}$＝"L"になるまでの遅延（入出力サイクル）		90		75	ns	C_L=50pF
	$t_{DL\overline{\Phi}}$ (IR)	クロックの立ち下がりから$\overline{IORQ}$＝"L"になるまでの遅延（INTAサイクル）		110		85	ns	
	$t_{DH\Phi}$ (IR)	クロックの立ち上がりから$\overline{IORQ}$＝"H"になるまでの遅延（INTAサイクル）		100		85	ns	
	$t_{DH\overline{\Phi}}$ (IR)	クロックの立ち下がりから$\overline{IORQ}$＝"H"になるまでの遅延（入出力サイクル）		110		85	ns	
$\overline{RD}$	$t_{DL\Phi}$ (RD)	クロックの立ち上がりから$\overline{RD}$＝"L"になるまでの遅延（入出力サイクル）		100		85	ns	C_L=50pF
	$t_{DL\overline{\Phi}}$ (RD)	クロックの立ち下がりから$\overline{RD}$＝"L"になるまでの遅延（メモリ・サイクル）		130		95	ns	
	$t_{DH\Phi}$ (RD)	クロックの立ち上がりから$\overline{RD}$＝"H"になるまでの遅延（M1サイクル）		100		85	ns	
	$t_{DH\overline{\Phi}}$ (RD)	クロックの立ち下がりから$\overline{RD}$＝"H"になるまでの遅延（M2～M5サイクル）		110		85	ns	
$\overline{WR}$	$t_{DL\Phi}$ (WR)	クロックの立ち上がりから$\overline{WR}$＝"L"になるまでの遅延（入出力サイクル）		80		65	ns	C_L=50pF
	$t_{DL\overline{\Phi}}$ (WR)	クロックの立ち下がりから$\overline{WR}$＝"L"になるまでの遅延（メモリ・サイクル）		90		80	ns	
	$t_{DH\overline{\Phi}}$ (WR)	クロックの立ち下がりから$\overline{WR}$＝"H"になるまでの遅延		100		80	ns	
	$t_W(\overline{WR_L})$	$\overline{WR}$のパルス幅("L")	〔10〕		〔10〕		ns	
$\overline{M1}$	t_{DL}(M1)	クロックの立ち上がりから$\overline{M1}$＝"L"になるまでの遅延		130		100	ns	C_L=50pF
	t_{DH}(M1)	クロックの立ち上がりから$\overline{M1}$＝"H"になるまでの遅延		130		100	ns	
$\overline{RFSH}$	t_{DL}(RF)	クロックの立ち上がりから$\overline{RFSH}$＝"L"になるまでの遅延		180		130	ns	C_L=50pF

信号	記号	パラメータ	Z-80 CPU 最小値	Z-80 CPU 最大値	Z-80A CPU 最小値	Z-80A CPU 最大値	単位	測定条件
$\overline{RFSH}$	$t_{DH}(RF)$	クロックの立ち上がりから$\overline{RFSH}$＝"H"になるまでの遅延		150		120	ns	C_L＝50pF
$\overline{WAIT}$	t_S (WT)	クロックの立ち下がりに対するセットアップ時間	70		70		ns	
$\overline{HALT}$	t_D (HT)	クロックの立ち下がりからの遅延		300		300	ns	C_L＝50pF
$\overline{INT}$	t_S (IT)	クロックの立ち上がりに対するセットアップ時間	80		80		ns	
$\overline{NMI}$	$t_W(\overline{NM_L})$	$\overline{NMI}$のパルス幅("L")	80		80		ns	
$\overline{BUSRQ}$	t_S (BQ)	クロックの立ち上がりに対するセットアップ時間	80		50		ns	
$\overline{BUSAK}$	$t_{DL}(BA)$	クロックの立ち上がりから$\overline{BUSAK}$＝"L"になるまでの遅延		120		100	ns	C_L＝50pF
	$t_{DH}(BA)$	クロックの立ち下がりから$\overline{BUSAK}$＝"H"になるまでの遅延		110		100	ns	
$\overline{RESET}$	t_S (RS)	クロックの立ち上がりに対するセットアップ時間	90		60		ns	
	t_F (C)	フロート状態になるまでの遅延（$\overline{MREQ}$，$\overline{IORQ}$，$\overline{RD}$および$\overline{WR}$）		100		80	ns	
	t_{mr}	$\overline{IORQ}$に先立つ$\overline{M1}$出力("L")の確定時間（INTAサイクル）	〔11〕		〔11〕		ns	

注〔1〕 $t_{acm}=t_W(\Phi_H)+t_f-75$
〔2〕 $t_{aci}=t_C-80$
〔3〕 $t_{ca}=t_W(\Phi_L)+t_r-40$
〔4〕 $t_{caf}=t_W(\Phi_L)+t_r-60$
〔5〕 $t_{dcm}=t_C-210$
〔6〕 $t_{dci}=t_W(\Phi_L)+t_r-210$
〔7〕 $t_{cdf}=t_W(\Phi_L)+t_r-80$
〔8〕 $t_W(\overline{MR_L})=t_C-40$
〔9〕 $t_W(\overline{MR_H})=t_W(\Phi_H)+t_f-30$
〔10〕 $t_W(\overline{WR_L})=t_C-40$
〔11〕 $t_{mr}=2t_C+t_W(\Phi_H)t_f-80$

○データを$\overline{RD}$に同期してバスに送り出すことが望ましい。割り込みアクノリッジ・サイクルでは$\overline{M1}$および$\overline{IORQ}$の両方に同期して送り出すことが望ましい。

○制御信号はすべて内部で同期がとれているため、クロックについて非同期的に使用してもよい。

○Ta＝+70℃、V_{CC}＝+5V±5%における負荷容量と出力の遅延との関係は次のとおりです。
負荷容量の50pF増加につき遅延は10ns増加します。負荷容量の最大値は、データ・バスが200pFで、他は100pFです。

○$\overline{RESET}$の入力幅は最低3クロック・サイクル必要です。

出力端子測定回路

AC タイミング図

"H"
"L"
CLOCK
OUTPUT
INPUT
FLOAT
Vcc-0.6V
2.0V
2.0V
ΔV
0.45V
0.8V
0.8V
±0.5V
Φ
A0-A15
A0-15
D0-7
IN
OUT
M1
RFSH
MREQ
RD
WR
IORQ
RD
WR
WAIT
HALT
INT
NM1
BUSRQ
BUSAK
RESET

（２） PIO

絶対最大定格

項　　目	記 号	定 格 値	単 位
入力電圧	V_{IN}	−0.3 〜 +7	V
出力電圧	V_{OUT}	−0.3 〜 +7	V
動作温度	T_{opr}	0 〜 +70	℃
保存温度	T_{stg}	−65 〜 +150	℃

電気的特性

DC 特性　　(Ta ＝0℃〜＋ 70 ℃ , V_{CC}＝＋5 V ± 5 %)

記 号	項　　目	最 小 値	最 大 値	単位	測 定 条 件
V_{ILC}	クロック“L”入力電圧	−0.3	0.45	V	
V_{IHC}	クロック“H”入力電圧	V_{CC} −0.6	V_{CC} +0.3	V	
V_{IL}	“ L ”入力電圧	−0.3	0.8	V	
V_{IH}	“ H ”入力電圧	2.0	V_{CC}	V	
V_{OL}	“ L ”出力電圧		0.4	V	I_{OL}＝ 2mA
V_{OH}	“ H ”出力電圧	2.4		V	I_{OH}＝−250μA
I_{CC}	消費電流		70	mA	
I_{LI}	入力リーク電流		10	μA	V_{IN}＝0〜V_{CC}
I_{LOH}	トライステート出力リーク電流		10	μA	V_{OUT}=2.4V〜V_{CC}
I_{LOL}	トライステート出力リーク電流		−10	μA	V_{OUT}＝0.4 V
I_{LD}	入力時のデータ・バスのリーク電流		±10	μA	$0 \leqq V_{IN} \leqq V_{CC}$
I_{OHD}	ダーリントン駆動電流	−1.5		mA	V_{OH}=1.5V ポートBのみ

端子容量

(Ta ＝+ 25 ℃ , f ＝1 MHz)

記 号	項　　目	最大値	単 位	測 定 条 件
C_{Φ}	クロック入力容量	12	pF	被測定端子以外のすべての端子は接地
C_{IN}	入力容量	7	pF	
C_{OUT}	出力容量	10	pF	

AC 特性

（T_a＝0℃～＋70℃、V_{CC}＝＋5V±5%）

信 号	記 号	パ　ラ　メ　ー　タ	Z-80 PIO 最小値	Z-80 PIO 最大値	Z-80A PIO 最小値	Z-80A PIO 最大値	単位	測定条件
Φ	t_C	クロック周期	400	〔1〕	250	〔1〕	ns	
	t_W（ΦH）	クロック・パルス幅（"H"）	170	2000	105	2000	ns	
	t_W（ΦL）	クロック・パルス幅（"L"）	170	2000	105	2000	ns	
	t_r, t_f	クロック立ち上がり・立ち下がり時間		30		30	ns	
	t_H	ホールド時間	0		0		ns	
$\overline{CE}$, C/D, B/A	$t_{S\Phi}$ (CS)	読み出しまたは書き込みサイクルの制御信号のセットアップ時間	280		145		ns	
D_0-D_7	t_{DR} (D)	$\overline{RD}$の立ち下がりからデータ出力までの遅延		430 〔2〕		380 〔2〕	ns	
	$t_{S\Phi}$ (D)	書き込みまたはM1サイクルのデータのセットアップ時間	50		50		ns	C_L＝50pF
	t_{DI} (D)	INTAサイクルの$\overline{IORQ}$の立ち下がりからデータ出力までの遅延		340 〔2〕		250 〔2〕	ns	
	t_F (D)	$\overline{RD}$または$\overline{IORQ}$の立ち上がりから出力バッファ・フロートまでの遅延		160		110	ns	
IEI	t_S (IEI)	INTAサイクルの$\overline{IORQ}$の立ち下がりに対するセットアップ時間	140		140		ns	
IEO	t_{DH} (IO)	IEIの立ち上がりからの遅延		210 〔4〕		160 〔4〕	ns	
	t_{DL} (IO)	IEIの立ち下がりからの遅延（注1）		190 〔4〕		130 〔4〕	ns	C_L＝50pF
	t_{DM} (IO)	$\overline{M1}$の立ち下がりからの遅延（M1サイクルの直前で割り込みが発生したとき）		300 〔4〕		190 〔4〕	ns	
$\overline{IORQ}$	$t_{S\Phi}$ (IR)	読み出しまたは書き込みサイクルのセットアップ時間	250		115		ns	
$\overline{M1}$	$t_{S\Phi}$ (M1)	INTAまたはM1サイクルのセットアップ時間	210		90		ns	
$\overline{RD}$	$t_{S\Phi}$ (RD)	読み出しまたはM1サイクルのセットアップ時間	240		115		ns	
$\overline{INT}$	t_D (IT)	$\overline{STB}$の立ち上がりからの遅延		490		440	ns	
	t_D (IT_3)	モード3のときのデーター致からの遅延		420		380	ns	

信号	記号	パラメータ	Z-80 PIO 最小値	Z-80 PIO 最大値	Z-80A PIO 最小値	Z-80A PIO 最大値	単位	測定条件
A_0-A_7, B_0-B_7	t_S (PD)	モード1のときの$\overline{STB}$の立ち上がりに対するセットアップ時間	260		230		ns	C_L=50pF
	t_{DS} (PD)	モード2のときの$\overline{STB}$の立ち下がりに対するセットアップ時間		230〔4〕		210〔4〕	ns	
	t_F (PD)	モード2のときの$\overline{STB}$の立ち上がりからポート・バス・フロートまでの遅延		200		180	ns	
	t_{DI} (PD)	モード0のときの書き込みサイクルの$\overline{IORQ}$の立ち上がりからポート出力確定までの遅延		200〔4〕		180〔4〕	ns	
$\overline{A\ STB}$, $\overline{B\ STB}$	t_W(ST)	$\overline{STB}$のパルス幅("L")	150〔3〕		150〔3〕		ns	
A RDY,	t_{DH} (RY)	$\overline{IORQ}$の立ち上がりからの応答時間		t_c+460〔4〕		t_c+410〔4〕	ns	C_L=50pF
B RDY	t_{DL} (RY)	$\overline{STB}$の立ち上がりからの応答時間		t_c+400〔4〕		t_c+360〔4〕	ns	

注　〔1〕 $t_C=t_W(\Phi H)+t_W(\Phi L)+t_r+t_f$

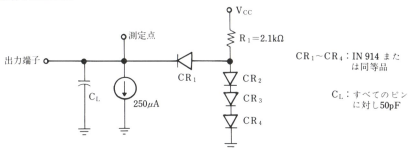
〔2〕 負荷容量の50pF増加につき、遅延は10ns増加します。 負荷容量の最大値は200pFです。

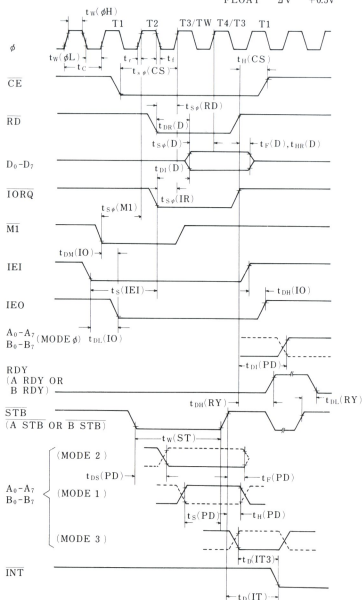
〔3〕 モード2のときは、$t_W(ST)>t_S(PD)$となります。

〔4〕 負荷容量の10pF増加につき、遅延は2ns増加します。 負荷容量の最大値は100pFです。

〔注1〕 デージー・チェーンがN段ある場合

$2.5t_C>(N-2)t_{DL}(IO)+t_{DM}(IO)+t_S(IEI)$+TTLバッファー遅延を満たさなければなりません。

出力端子測定回路

ACタイミング図

"H" "L"
CLOCK 4.2V 0.8V
OUTPUT 2.0V 0.8V
INPUT 2.0V 0.8V
FLOAT ⊿V +0.5V
tW(φH)
T1
T2
T3/TW
T4/T3
T1
φ
tW(φL)
tC
tr
tf
tsφ(CS)
tH(CS)
CE
tsφ(RD)
RD
tDR(D)
tsφ(D)
tF(D), tHR(D)
D0-D7
tDI(D)
IORQ
tsφ(IR)
tsφ(M1)
M1
tDM(IO)
IEI
ts(IEI)
tDH(IO)
IEO
A0-A7
B0-B7
(MODE φ)
tDL(IO)
tDI(PD)
RDY
(A RDY OR
B RDY)
tDH(RY)
tDL(RY)
STB
(A STB OR B STB)
tW(ST)
(MODE 2)
tDS(PD)
tF(PD)
A0-A7
B0-B7
(MODE 1)
ts(PD)
tH(PD)
(MODE 3)
tD(IT3)
INT
tD(IT)

（3） CTC

絶対最大定格

項　目	記 号	定　格	単位
入 力 電 圧	V_{IN}	−0.3〜＋ 7	V
出 力 電 圧	V_{OUT}	−0.3〜＋ 7	V
動 作 温 度	T_{opr}	0〜＋ 70	℃
保 存 温 度	T_{stg}	−65〜＋150	℃

電気的特性

DC 特性　（T_a＝0℃〜＋70℃，V_{CC}＝＋5V±5％）

記 号	項　目	最 小 値	最 大 値	単位	測 定 条 件
V_{ILC}	クロック“L”入力電圧	−0.3	0.45	V	
V_{IHC}	クロック“H”入力電圧	V_{CC}−0.6	V_{CC}＋0.3	V	
V_{IL}	“L”入力電圧	−0.3	0.8	V	
V_{IH}	“H”入力電圧	2.0	V_{CC}	V	
V_{OL}	“L”出力電圧		0.4	V	I_{OL}＝2mA
V_{OH}	“H”出力電圧	2.4		V	I_{OH}＝−250μA
I_{CC}	消費電流		120	mA	t_C＝400ns
I_{LI}	入力リーク電流		10	μA	V_{IN}＝0V〜V_{CC}
I_{LOH}	トライステート出力リーク電流		10*	μA	V_{OUT}＝2.4V〜V_{CC}
I_{LOL}	トライステート出力リーク電流		−10*	μA	V_{OUT}＝0.4V
I_{OHD}	ダーリントン駆動電流	−1.5*		mA	V_{OH}＝1.5V ZC／TO_0〜ZC／TO_2 に適用

＊流入電流を正、流出電流を負とします。

端子容量　（T_a＝＋25℃，f＝1MHz）

記 号	項　目	最大値	単 位	測 定 条 件
CΦ	クロック入力容量	25	pF	被測定端子以外のすべての端子は接地
C_{IN}	入 力 容 量	5	pF	
C_{OUT}	出 力 容 量	10	pF	

AC 特性

（T_a＝0℃～＋70℃、V_{CC}＝＋5V±5％）

信号	記号	パラメータ	Z-80 CTC 最小値	Z-80 CTC 最大値	Z-80A CTC 最小値	Z-80A CTC 最大値	単位	備考
Φ	t_C	クロック周期	400	〔1〕	250	〔1〕	ns	
	t_W(ΦH)	クロック・パルス幅("H")	170	2000	105	2000	ns	
	t_W(ΦL)	クロック・パルス幅("L")	170	2000	105	2000	ns	
	t_r, t_f	クロック立ち上がり・立ち下がり時間		30		30	ns	
	t_H	ホールド時間	0		0		ns	
CS, $\overline{CE}$	$t_{S\Phi}$(CS)	読み出し、または書き込みサイクルの制御信号のセットアップ時間	160		60		ns	
D_0-D_7	t_{DR}(D)	$\overline{RD}$の立ち下がりからデータ出力までの遅延		480		380	ns	〔2〕
	$t_{S\Phi}$(D)	書き込み、またはM1サイクルのデータのセットアップ時間	60		50		ns	
	t_{DI}(D)	INTAサイクルの$\overline{IORQ}$の立ち下がりからデータ出力までの遅延		340		160	ns	〔2〕
	t_F(D)	$\overline{RD}$の立ち上がりから出力バッファ・フロートまでの遅延		230		110	ns	
IEI	t_S(IEI)	INTAサイクルの$\overline{IORQ}$の立ち下がりに対するセットアップ時間	200		140		ns	
IEO	t_{DH}(IO)	IEIの立ち上がりからの遅延		220		160	ns	〔3〕
	t_{DL}(IO)	IEIの立ち下がりからの遅延		190		130	ns	〔3〕
	t_{DM}(IO)	$\overline{M1}$の立ち下がりからの遅延（M1サイクルの直前で割り込みが発生したとき）		300		190	ns	〔3〕
$\overline{IORQ}$	$t_{S\Phi}$(IR)	読み出し、または書き込みサイクルのセットアップ時間	250		115		ns	
$\overline{M1}$	$t_{S\Phi}$(M1)	INTA、またはM1サイクルのセットアップ時間	210		90		ns	
$\overline{RD}$	$t_{S\Phi}$(RD)	読み出し、またはM1サイクルのセットアップ時間	240		115		ns	
$\overline{INT}$	t_{DCK}(IT)	CLK/TRGの立ち上がりからの遅延		$2t_C$(Φ)＋200		$2t_C$(Φ)＋140	ns	カウンタ・モード
	$t_{D\Phi}$(IT)	Φの立ち上がりからの遅延		t_C(Φ)＋200		$2t_C$(Φ)＋140	ns	タイマ・モード
CLK/TRG 0−3	t_C(CK)	カウンタ・クロック周期	$2t_C$(Φ)		$2t_C$(Φ)		ns	カウンタ・モード
	t_r(CK/TR) t_f(CK/TR)	カウンタ・クロックおよびトリガの立ち上がり・立ち下がり時間		50		30	ns	
	t_S(CK)	即時カウントに要するクロックのセットアップ時間	210		130		ns	カウンタ・モード
	t_S(TR)	プリスケーラの即時起動に要するトリガのセットアップ時間	210		130		ns	タイマ・モード
	t_W(CTH)	カウンタ・クロックおよびトリガのパルス幅("H")	200		120		ns	カウンタ・モードおよびタイマ・モード
	t_W(CTL)	カウンタ・クロックおよびトリガのパルス幅("L")	200		120		ns	
ZC/TO 0−2	t_{DH}(ZC)	Φの立ち上がりからZC/TO="H"までの遅延		190		120	ns	カウンタ・モードおよびタイマ・モード
	t_{DL}(ZC)	Φの立ち下がりからZC/TO="L"までの遅延		190		120	ns	

注〔1〕 $t_C = t_W(\Phi_H) + t_W(\Phi_L) + t_r + t_f$

〔2〕 負荷容量の50pF増加につき、遅延は10ns増加します。負荷容量の最大値は、データ・バスが200pFであり、他は100pFです。

〔3〕 負荷容量の10pF増加につき遅延は2ns増加します。負荷容量の最大値は100pFです。

〔4〕 $\overline{\text{RESET}}$の入力幅は最低3クロック・サイクル必要です。

出力端子測定回路

AC タイミング図

"H"
"L"
CLOCK
Vcc-0.6V
0.45V
OUTPUT
2.0V
0.8 V
INPUT
2.0V
0.8V
FLOAT
ΔV
±0.5V
tw(ΦH)
T1
T2
T3/TW
T4/T3
T1
Φ
tw(ΦL)
tr
tf
tc
ts φ(CS)
tH(CS)
CE
ts φ(RD)
RD
tDR(D)
ts φ(D)
tF(D), tHR(D)
D0-D7
tDI(D)
ts φ(IR)
IORQ
ts φ(M1)
M1
tDM(IO)
IEI
ts(IEI)
tDH(IO)
IEO
tDL(IO)
tD φ(IT)
INT
tc(CK)
tDCK(IT)
ts(CK)
tw(CTH)
(COUNTER
MODE)
tf(CK)
CLK/
TRG0-3
tr(CK)
tw(CTL)
ts(TR)
tf(TR)
tw(CTH)
(TIMER
MODE)
tDH(ZC)
tr(TR)
tw(CTL)
ZC/TO0-2
tDL(ZC)

索　　引

ワ　行

数字、アルファベット

〈著者略歴〉
横田英一（よこた　えいいち）
昭和 47 年　東京電機大学工学部
電子工学科卒
現　在　シャープ株式会社
電子部品営業本部

新版 図解 Z-80 の使い方

平成 5 年 8 月 20 日　第 1 版第 1 刷発行
平成 15 年 3 月 15 日　第 1 版第 14 刷発行

著　者　横田英一
発行者　佐藤政次
発行所　株式会社 オーム社
郵便番号　101-8460
東京都千代田区神田錦町 3-1
電 話 03(3233)0641（代表）
URL http://www.ohmsha.co.jp/

印刷　中央印刷　　製本　三水舎
ISBN4-274-07759-4 Printed in Japan

わかる本のご案内

「わかる本」は,いま必要な技術と知識をテーマに,その本質を図やイラストを用いて,やさしく・わかりやすくをモットーに編集した入門書です.専門学校生,大学生から社会人の方々の必読の書です.

Javaがわかる本

イーズ・コミュニケーションズ株式会社 編
(A5判・128頁・本体 1500円)

目次 Javaの概要/Javaの歴史/言語としてのJava/Javaを用いたユーザインタフェース/組込み向けのJava/エンタープライズ向けのJava/JavaとXML

ネットワークがわかる本

石川 裕 著
(A5判・132頁・本体 1500円)

目次 コンピュータシステムとネットワーク/ネットワークの伝送技術, 交換技術/プロトコル/LAN/WAN/インターネット/通信のセキュリティ/これからのネットワーク

通信プロトコルがわかる本

石川 裕 著
(A5判・142頁・本体 1500円)

目次 データ通信の基礎/下位層のプロトコル/上位層のプロトコル/LANとTCP/IPのプロトコル/実際のネットワークとプロトコル/今後のネットワーク

インターネットがわかる本

芝野耕司 著
(A5判・148頁・本体 1500円)

目次 インターネットとは/インターネットの現在/インターネットの仕組み/電子メールとその拡張/WWWの仕組み/インターネットの未来

Webサイトがわかる本

樫山友一 著
(A5判・128頁・本体 1500円)

目次 Webサイトとは/Webサイトの開発プロセス/Webサイトの内部/セキュリティ方式の決定/コンピュータ構成の決定/Webサイトの設計/Webサイトのテスト/Webサイトの運用

SGML/XMLがわかる本

芝野耕司 著
(A5判・162頁・本体 1500円)

目次 SGMLへ至る道/標準一般化マーク付け言語SGML/SGMLからHTML, そしてXML/形式適合XML文書/正しいXML文書/XML Names—XMLの名前空間/XPath—XMLでの部分文書の指定/XSLT—XMLでのスタイル指定と変換/XMLをうまく使いこなすために

データベースがわかる本

鈴木健司 著
(A5判・120頁・本体 1500円)

目次 データベースの概念/データモデルの概念/関係データモデル/データベースの設計/データベース言語SQL/関係DBMSのGUI/データベースの新たな展開

オブジェクト指向がわかる本

佐藤英人 著
(A5判・136頁・本体 1600円)

目次 オブジェクト指向の基礎/オブジェクトの実装/オブジェクト指向プログラミング/オブジェクトの部品化と再利用/オブジェクトの分析と設計/オブジェクト指向の展開

SQLがわかる本

芝野耕司 著
(A5判・136頁・本体 1400円)

目次 リレーショナルデータベースとSQL/表とビューの定義—データ定義言語(DDL)—/SQLでの問合せとMSAccessのクエリー/表の統合(JOIN)と併合(UNION)/SQLでのデータ操作/Accessのフォームとレポート/副問合せ/データベース設計と正規化/カーソル処理とホスト言語からの呼び出し/トランザクション管理と安全保護/SQLの拡張

データウェアハウスがわかる本

鈴木健司 著
(A5判・120頁・本体 1500円)

目次 データウェアハウスの概念/データウェアハウスの基本構成/データウェアハウスの設計と構築/多次元データウェアハウス/データマートの設計と構築/データウェアハウスの活用/データウェアハウス構築の開発手順

プログラミングがわかる本

本田哲夫 著
(A5判・152頁・本体 1500円)

目次 コンピュータ言語とは/コンピュータ言語の第一歩/BASIC言語でプログラミング/C言語プログラミング/オブジェクト指向プログラミング/Visual Basicで作成/応用例題と実力アップ

オーム社/出版局

ISBN4-274-07759-4
C3055 ¥2800E
9784274077593

定価(本体2800円【税別】)

新版 図解

Z-80の使い方